Hans Dieter Winte

Analyse des Commodore-BASIC 4.0 und BASIC 3.0

MIKRO + KLEIN COMPUTE

Analyse des Commodore-BASIC 4.0 und BASIC 3.0

Hans Dieter Winter

Analyse des Commodore-BASIC 4.0 und BASIC 3.0

MIKRO +KLEIN COMPUTER

Die Informationen im vorliegenden Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit grösster Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag und Autor können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Autor dankbar.

ISBN 3-907007-01-8

© 1982 by Mikro+Kleincomputer Informa Verlag AG, CH-6000 Luzern 15
Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlages reproduziert oder unter Verwendung elektronischer Systeme vervielfältigt oder verbreitet werden, ausgenommen Speicherung in Datenverarbeitungsanlagen für den eigenen Gebrauch.
Einbandgestaltung: Hans-Jürgen Ottenbacher
Druck: Unionsdruckerei Luzern / Printed in Switzerland

Inhaltsverzeichnis

Einleitung	7
Notationen	8
Allgemeine Aufgaben eines Betriebssystems	11
A. Initialisierung	12
B. Monitor	13
C. I/O Management	14
D. Speicherplatzverwaltung	16
Das Betriebssystem des CBM-Rechners	17
Initialisierung	18
Ablauf der Initialisierung	21
Verwendbare Routinen	22
Der TIM-Monitor	24
Die Arbeit mit dem TIM	26
Die Monitor-Warteschleife	28
Befehle des Monitors	29
Die ERROR-Routine	33
Eigenarten des Monitors	33
Nützliche Hilfen für die Programmierung in Maschinensprache	34
Einbau der IRO, NMI, RES-Tasten	35
Anwendung der Tasten	36
Der Stack des CBM	38
Speicherverwaltung	47
Speicheraufteilung	47
Speicherablage der verschiedenen Datentypen	50
Ablage eines BASIC-Programms	50
Kennzeichnung von Variablennamen	52
Ablage von einfachen Variablen	53
Ablage von indizierten Variablen	54
Ablage der Stringinhalte	59
Besonderheiten bei Stringvariablen	61
Verwendbare Routinen	62
Ein/Ausgabe	75
Schnittstellen eines Rechners	76
Der CBM-IEC-Bus	80
Ablauf des Handshakes	80
Management-Leitungen	81
Statusmeldungen	82
Konversation mit dem IEC-Bus (prinzipiell)	83
IEC-Bus-Routinen (verwendbar)	85
IEC-Bus-Steuerung in Maschinensprache	85
Ausgabe über den IEC-Bus	92
Ausgabe auf den Bildschirm	97

Meldungen des Betriebssystems 105
Eingabe über den IEC-Bus 107
Eingabe von Tastatur und Bildschirm 110
Der Tastaturpuffer 110
Eingaberoutinen 111
Ein/Ausgabe mit Kassettenrekorder 116
Ausgabeformat und Speicherbelegung 116
Kassettenrekorderoperationen 120
Disk-Handling im BASIC 4.0 126
Der CBM-User-Port 132
Die Registeradressen des VIA 6522 133
Beispiele zum User-Port 134

Mathematik 136
Zahlendarstellung 136
Ganze Zahlen 137
Fliesskommazahlen 137
Verwendbare Routinen 140

BASIC-Interpretation 161
Prinzipielle Vorgehensweise 161
Umwandlung in den BASIC-Code 162
Interpretation der BASIC-Befehle 166
Verwendbare Routinen 169

Erweiterung des Betriebssystems 171

Anhang
Kontrolle des IEC-Busses 178
Ausgabe auf den IEC-Bus 179
Ausgabe auf den Bildschirm 180
Meldungen 184
Bildschirmcodes 185
Bildschirmpeicherplan 186
Eingabe vom IEC-Bus 187
Tastatur/Bildschirm-Eingabe 188
Ein/Ausgabe Kassettenrekorder 189
Überprüfung von Tasten 190
Disketten-Management 191
Register-Transfers 192
Zahlen-Umwandlungen 194
Arithmetische Routinen 195
Monadische Operationen 197
Prüf-Routinen 198
Konstanten im ROM 199
Zahlendarstellung 200
Suchroutinen 202
Tokens 203
Anschlussbelegung der CBM-Ports 205
Systemadressen 206

Einleitung

Die Kleinrechner haben in den letzten Jahren immer mehr an Bedeutung gewonnen, nicht nur im Hobbybereich, sondern zunehmend auch im kommerziellen Bereich.
Die Innovationen und Verkaufstendenzen auf diesem Gebiet sind weiterhin steigend.

Schaltet man die Stromversorgung eines solchen Kleinrechners ein, so ist damit ein routinemässiges Arbeiten mit dem System ohne eine bestimmte Menge an Software nicht möglich. Diese Software bildet das Betriebssystem des Rechners.
Bei der Betrachtung solcher Betriebssysteme zeigt es sich, dass ein einfacher Einkartencomputer für die bei ihm anfallenden Operationen ähnliche Grundaufgaben erfüllen muss wie ein Grosscomputer eines Timesharingsystems im entsprechenden Einsatzbereich. Mit dem Fortschreiten der Entwicklung werden auch im Microcomputerbereich die Betriebssysteme immer komplexer.

Jeder Kleinrechner besitzt heute schon eine der höheren Programmiersprachen wie z.B. BASIC, PASCAL etc., die es dem Anwender abnehmen, auf Maschinensprachenebene relativ kompliziert mit dem System zu korrespondieren. Unter diesen höheren Programmiersprachen, die einen grossen Teil des Betriebssystemes darstellen, ist immer noch BASIC am weitesten verbreitet, da es gerade für den Anfänger als leicht erlernbare Programmiersprache besonders geeignet ist. In Zukunft werden sich hier jedoch auch Pascal und evtl. Forth, wie auch CP/M (bei 16-Bit-Rechnern und Rechnern mit 8085- oder Z80-CPU) weiter durchsetzen.

Die Geräte der Firma Commodore sind in Deutschland unter den am weitesten verbreiteten Microcomputern zu finden.

Die vorliegende Abhandlung baut auf Veröffentlichungen (siehe Literatur (1) bis (12)) auf, um das gesamte Betriebssystem der CBM-Rechner zu analysieren.
Dazu wird nach einführenden Erklärungen immer tiefer in das System "eingestiegen".
Der Anhang in Form eines "Kochbuches" ermöglicht die rezeptartige Verwendung der Routinen des Betriebssystems für die komfortable Erstellung eigener Maschinensprache-Programme.

Kenntnisse des Befehlssatzes der CPU 6502 müssen dafür vorausgesetzt werden.

Notationen

I. Abkürzungen

Folgende Abkürzungen werden verwendet:

AC	=	Akku
XR	=	X-Register
YR	=	Y-Register
SR	=	Statusregister
SP	=	Stackpointer
PC	=	Programmzähler
LB	=	Low-Byte
HB	=	High-Byte
1234	=	Dezimale Adressangabe
$1234	=	Hexadez. Adressangabe
(LB,HB)	=	Adresse LB,HB
(LB/HB)	=	Inhalt der Adresse LB,HB
(XX)	=	Inhalt der Zeropageadresse XX bzw. des Registers XX
Anhang XX	=	Anhang Seite XX

Sind zwei Zahlenwerte durch ein '<' getrennt angegeben, so ist der erste gültig für die CBM 3000-Serie, der zweite für die CBM 8000-Serie.
Bei jeder angegebenen Adresse sind die LABELS angegeben, wie sie auch in der bisher bestehenden Literatur (siehe Anhang) verwendet werden.
Außerdem sind beim Verfasser symbolische Assemblerlistings der COMMODORE-Betriebssysteme erhältlich, die ebenfalls diese LABELS verwenden, sowie eine Diskette mit diesem Listing als Bibliothek für einen Macro-Assembler. Systemadressen können damit symbolisch verwendet werden, da deren Absolut- Adresse vom Assembler in der Bibliothek gesucht werden kann.

II. Hinweise zu den Assemblerbeispielen

Fast alle aufgeführten Assemblerbeispiele sind sowohl vom Monitor mit .G 033A als auch von BASIC mit SYS 826 startbar. Die meisten der angegebenen Assemblerbeispiele sind für einen Start vom Monitor aus vorgesehen. Um diese Programme mit SYS XXX aufrufen zu können, muss nur jeweils der Maschinensprachebefehl "BRK" (Sprung in den TIM) am Ende eines Programmes durch ein "RTS" ersetzt werden.
Wenn bei einem Assemblerbeispiel ein zugehöriges BASIC-Programm angegeben ist, so muss dieses einmal mit RUN gestartet worden sein, bevor das dazugehörige Maschinenprogramm vom Monitor aus mit G XXXX aufgerufen werden kann. Bei einem Aufruf von BASIC mit SYS XXX entfällt dies natürlich.

Die Beispiele sind grundsätzlich für den CBM 3000 ausgeführt, das Hexlisting ist daher nur auf diesem lauffähig!
Für den CBM 8000 muss der in Klammern angegebene Wert im symbolischen Listing eingesetzt werden.

III. Anhang

Im Anhang sind die verwendbaren Routinen in "Rezeptform" aufgeführt.
Die Tabellen gliedern sich auf in 4 Spalten, die folgende Bedeutungen haben:

Die erste Spalte zeigt den LABEL zu der entsprechenden Routine. Diese LABEL werden größtenteils in der bisher erschienenen Literatur zu den COMMODORE-Rechnern verwendet.
In der zweiten Spalte steht die Einsprungsadresse für BASIC 3.0, in der dritten die Adresse für BASIC 4.0.
In der vierten Spalte sind die nötigen Voraussetzungen (unter E:) und die resultierenden Ausgaben bzw. Wirkungen (unter A:) aufgeführt.
Die fünfte Spalte zeigt die Register, die durch die Routinen verändert werden.
Die Bedeutung der jeweiligen Routine ist in der letzten Spalte angegeben.

IV. Erklärungen zum Assembler

Die Beispiele in Maschinensprache werden auf den ersten Blick vielleicht etwas verwirren, da darin ungewohnte Befehle und Zeichen auftreten.
Um sich leichter hindurchzufinden, sind an dieser Stelle die gebrauchten Definitionen aufgeführt.
Mit etwas Eingewöhnungszeit wird man sich schnell in den Listings zurechtfinden, die es ermöglichen, die Programme sowohl hexadezimal mit dem TIM als auch symbolisch mit einem beliebigen Assembler einzugeben.

Das Listing besteht aus 6 Spalten:

1. Die 1.Spalte gibt die hexadezimale Adresse an, ab welcher der Befehl in der jeweiligen Zeile beginnt.
2. In der 2.Spalte befinden sich die hexadezimalen Befehlscodes. Adressen sind schon (folgerichtig) in der Reihenfolge LB,HB angegeben
3. Die Zeilennummer in der 3.Spalte hat keine Bedeutung für das Programm.
4. In der 4.Spalte befinden sich die symbolischen Labels, auf die meistens durch Verzweigungsbefehle gesprungen wird.
5. Die 5.Spalte beinhaltet die mnemonischen Codes sowie die Wert- und Adressen-Zuweisungen in hexadezimaler oder symbolischer Form.
6. Der Kommentar in der letzten Spalte hat keine Bedeutung für das Programm, sondern dient nur zur Erklärung desselben und wird durch ein ';' von den Befehlen getrennt.

Es bleiben noch folgende Zeichen zu erklären:

.BA gibt an, an welcher Stelle das Maschinenprogramm beginnen soll.
.DE definiert den davor angegebenen Label als die folgende Adresse oder Speicherstelle.
.BY bewirkt, dass die darauffolgenden Zeichen ab dieser Stelle im Speicher abgelegt werden sollen.
.EN kennzeichnet das physikalische Ende des Assemblerprogrammes.

Dies sind die sog. Pseudo-Operations-Codes, die den Assemblierlauf steuern.

* weist auf Zeropage-Adressierung hin.
\# ist eine direkte Wertzuweisung.
$ kennzeichnet hexadezimale Werte.
' kennzeichnet ASCII-Zeichen.
% kennzeichnet binäre Darstellungen.
Dezimale Werte werden ohne Erkennungszeichen notiert.

H,Label entspricht dem 'High-Byte von LABEL'
L,Label entspricht dem 'Low-Byte von LABEL'

Ein 'A' bedeutet Akkumulator-Adressierung.

Allgemeine Aufgaben eines Betriebssystems

Jedes Betriebssystem muss bestimmte Grundaufgaben ausführen können, um überhaupt ein lauffähiges Rechnersystem zu bilden.
Diese Grundaufgaben lassen sich in folgende vier übergeordnete Hauptaufgaben unterteilen:

A. Initialisierung

B. Monitor

C. I/O-Management

D. Speicherverwaltung

Ausser diesen vier Hauptaufgaben muss das Rechnersystem in der Praxis noch die vielfältigsten anderen Aufgaben erfüllen, die von Anwender zu Anwender verschieden sein können.
So kann beispielsweise der Komfort durch Einbinden einer höheren Programmiersprache wie BASIC, FORTRAN, PASCAL usw. erheblich gesteigert werden.
Ausserdem übernimmt das Betriebssystem oft auch die Steuerung von mehreren Peripheriegeräten für die Ein- und Ausgabe von Rohdaten und Ergebnissen.

Im folgenden sollen die vier obengenannten Hauptaufgaben allgemein betrachtet werden.

A. Initialisierung

Die Initialisierung eines Systems hat die Aufgabe, den Rechner nach dem Einschalten in einen bestimmten Anfangszustand zu versetzen und einen Einstieg in den Dialog zwischen Anwender und Rechner zu ermöglichen.
Die Initialisierungsroutinen sind also Programmteile des Betriebssystems, die ohne einen Anwendereingriff automatisch ablaufen.

Im Anfangszustand müssen die teilweise vielfältigen Funktionsregister und Flags sowohl der Zentraleinheit als auch der Peripheriebausteine, programmiert bzw. so gesetzt sein, dass bestimmte, berechenbare Funktionen der Bausteine erwartet werden können.

Die Aufgaben während der Initialisierung können sehr vielfältig sein.

Einige Beispiele daraus sollen hier genannt werden:

a. Flag- und Steuerwortregister, Programmstatus, Refreshregister der CPU usw. setzen.

b. Diverse Baudraten-Zähler setzen.

c. Ein- und Ausgänge bei Parallelschnittstellen definieren.

d. CRT-Steuerbausteine programmieren (Synchronisation, Zeilenbreite ...).

e. Speicherbereiche, Pointer und Softwareschalter auf die benötigten Anfangswerte setzen.

f. Interrupt-Vektoren laden.

Die Zeit, die die Initialisierungsphase benötigt, hängt massgeblich ab von der Grösse und der Komplexität des Rechnersystems.
Wird eine Datenverarbeitungsanlage eingeschaltet, so vollzieht sich diese Prozedur im Bereich von Mikrosekunden bei Einplatinencomputern bis zu einigen Sekunden bei grossen Systemen.

B. Monitor

Jeder Rechner - angefangen beim Einplatinencomputer bis hin zum kommerziellen Grossrechner - besitzt ein verhältnismässig primitives, aber dennoch ausserst nützliches und notwendiges Grundbetriebsprogramm: den MONITOR.

Dieses Betriebsprogramm erlaubt das Arbeiten mit dem Rechner auch ohne das Vorhandensein einer höheren Programmiersprache.
Man korrespondiert dann mit dem Rechner auf seiner untersten Ebene, der Maschinensprachebene. Hier ermöglichen einige Befehle z.B. das Ausgeben und Ändern von Speicherzellen.
Viele Grossrechner besitzen nach dem Einschalten nur ein solches - verhältnismässig winziges - Grundbetriebssystem, das es gestattet, das eigentliche Betriebssystem von einem externen Datenträger (Lochstreifen, Plattenspeicher) in den Arbeitsspeicher zu laden.
Erst dann kann komfortabel mit dem Rechner gearbeitet werden, um komplexe Programme zu erstellen oder Peripheriegeräte zu steuern.

Die vielen Kleincomputer, die heute auf dem Markt sind, haben zumeist eine höhere Programmiersprache fest im ROM installiert. Es ist also möglich, sofort nach dem Einschalten des Rechners z.B. in BASIC zu programmieren (CBM, TRS 80).
Aber auch in dem Bereich Microcomputer gibt es Geräte, die nur einen sog. Bootstraploader oder Urlader besitzen, um eine höhere Sprache von Kassette oder Diskette in den Arbeisspeicher zu laden (Eurocom).

C. I/O-Management

Der Computer wäre zu nichts nütze, wenn er nichts anderes könnte, als seinen Speicher zu verwalten, Rechenroutinen auszuführen und andere Dinge, die er ja bekanntlich sehr schnell auszuführen vermag.
Vielmehr muss er die Möglichkeit besitzen, mit Peripheriegeräten zu korrespondieren, welche er dann sehr schnell mittels seiner einprogrammierten Algorithmen und Routinen bedienen kann.

Um Daten oder Befehle erhalten zu koennen, bedient sich der Computer der vielfältigsten Geräte, die ihm den Kontakt mit der Umwelt auf fast jede erdenkliche Art und Weise erlauben:

Ueblicherweise holt er sich zumindest die ersten Befehle (z.B. beim Programmieren) von einer Tastatur und gibt seine Reaktion bei einfacheren Versionen auf ein LED-Display aus, währenddem die kostspieligeren Versionen die Ausgabe auf ein Video-Dispay ermöglichen.
Die Kombination von Bildschirm mit Tastatur nennt man auch Terminal.

Nachdem der Computer mit dieser Ausrüstung schon Befehle vom Programmierer empfangen kann, benötigt er noch Möglichkeiten, seine Ergebnisse oder Programme in resistenter Form auszugeben, indem er diese z.B. auf einen Drucker ausgibt, oder auf ein Speichermedium schreibt.

Zu diesen Speichermedien gehörten früher die schon fast legendären Kernspeicher (oft wird auch der Arbeitsspeicher des Computers fälschlich als Kernspeicher bezeichnet), die heute allerdings durchweg durch Halbleiterspeicher ersetzt worden sind.

Zur dauerhaften Abspeicherung verwendet der Computer Massenspeicher, die vom Kassettenrekorder über das Magnetband und die flexible Disketten bis hin zum Festplattenlaufwerk -mit entsprechend steigenden Kosten- reichen.

Im Heimcomputermarkt hat sich hier vorwiegend das billigste Medium, nämlich der Kassettenrekorder durchgesetzt und ist in den verschiedensten Ausführungen verbreitet: vom billigen Kaufhausrekorder (ca.60 DM) bis zum relativ schnellen Digitalrekorder (ca.400 DM).

Allerdings setzt sich hier immer mehr die verhältnismässig preiswerte, flexible Minidisk durch, deren Anschaffungspreis nicht unter 2000.-DM liegt. Diese Disketten-Drives können innerhalb kürzester Zeit auf einen Speicher von über 1 Megabytes zugreifen, was die Microcomputer auch immer attraktiver für den kommerziellen Anwender macht.

Seit einiger Zeit sind Winchesterdrives auf dem Markt (nicht unter 5000.-DM), die mit einer Speicherkapazität von mehreren -zig Megabyte die Lücke zwischen den Diskettendrives und den Festplatten-Speichern schliessen.
Diese Festplatten werden (wegen Ihres hohen Preises) in der Industrie oder bei Anwendern mit grossen Datenmengen eingesetzt.

Aber mit diesen Massenspeichern sind noch lange nicht die Grenzen des Informationsaustausches mit der Umwelt erreicht!
Der Computer kann über sog. Analog-Digital-Wandler fast alle analogen Grössen aufnehmen wie Spannung, Strom, Temperatur und alle physikalischen Grössen aus der Dynamik (Länge, Geschwindigkeit, Beschleunigung) sowie beinahe alle erdenklichen Messungen durchführen, die mit irgendwelchen Messgeräten aufgenommen werden können.

Die umgekehrte Richtung ist z.B. durch Digital-Analog- Wandler ebenfalls möglich:
Der Computer kann - neben den digitalen - auch analoge Signale ausgeben, womit ihm fast unbegrenzte Möglichkeiten gegeben werden, in Abläufe einzugreifen und komplizierte Steuerungen präzise auszuführen.

Um nun all diese Peripheriegeräte bedienen zu können benutzt der Computer einen bzw. mehrere Ausgänge, über die er mit ihnen korrespondieren kann.
Beim CBM-Computer sind dies:

2 serielle Schnittstellen für die Kassettenrekorder,
eine 8-Bit-parallele Schnittstelle mit Handshake
ein IEC- Bus zur Ansteuerung von bis zu 12 Standard-Peripheriegeräten.

D. Speicherverwaltung

Um veränderliche Daten aufnehmen zu können, besitzt jeder Rechner einen mehr oder weniger grossen Arbeitsspeicher.
Die Verwaltung dieses Speicherbereiches ist eine der grundlegenden Aufgaben eines Betriebssystems.

Aber nicht die Grösse des verfügbaren Arbeitsspeichers, sondern die optimale Nutzung des vorhandenen RAM-Bereiches und eine "ausgeklügelte" Aufteilung unter den verschiedenen Datentypen spielen hier die entscheidende Rolle.

Bei der Lösung dieser Probleme gibt es prinzipiell zwei Methoden der Speicherverwaltung:

a. Statische Speicherverwaltung
b. Dynamische Speicherverwaltung

Bei der statischen Speicherverwaltung sind für die verschiedenen, möglichen Datentypen feste Speicherbereiche reserviert, deren Grenzen nicht veränderbar sind. Diese Methode hat den Vorteil, dass die Verwaltung und der Zugriff auf die einzelnen Datenbereiche relativ unkompliziert und zudem sehr schnell ist.
Sie besitzt jedoch den grossen Nachteil, dass entweder viel Speicherplatz verschenkt wird, wenn z.B. bei dem speziellen Anwendungsfall bestimmte Datentypen nur in geringer Anzahl auftreten, oder dass der vorgesehene Speicherplatz für einen Datenbereich manchmal nicht ausreicht.

Dieser grosse Nachteil wird durch eine dynamische Speicherverwaltung völlig beseitigt.
Denn diese Methode passt die verschiedenen Datenbereiche laufend den anfallenden Datenmengen an, d.h. die einzelnen Datenbereiche schrumpfen zusammen oder wachsen auseinander je nach Anzahl der aufzunehmenden Daten.
Der Hauptvorteil liegt also darin, dass durch diese Methode der vorhandene Arbeitsspeicher bis auf das "letzte Byte" ausgenutzt werden kann.
Obwohl diese Art der Speicherverwaltung ein bedeutend komplexeres Management-Softwaresystem erfordert, wird sie heute am häufigsten angewendet.

In vielen Rechnersystemen sind auch beide Methoden der Speicherverwaltung gleichzeitig zu finden.

Das Betriebssystem des CBM

Nach diesen allgemeinen Erörterungen über die Aufgaben eines jeden Betriebssystemes sollen im speziellen die Betriebssysteme der Personal-Computer CBM 3000 und CBM 8000 der Firma COMMODORE untersucht werden.
Diese Rechner besitzen ein 14 kB umfassendes Betriebssystem, wovon einen Grossteil der BASIC-Interpreter von der Firma MICROSOFT mit 8 kB einnimmt.
Die Unterschiede zwischen den beiden Systemen liegen funktionsmäßig in der Art der Stringablage, dem Disk-Basic und dem CRT-Controller des CBM 8000.
Die Betriebssysteme sind softwaremässig mit dem Befehlsvorrat des Mikroprozessors 6502 realisiert.

Zu der nun folgenden Analyse wurden die Betriebssysteme in sechs Aufgabengebiete unterteilt:

1. I n i t i a l i s i e r u n g
2. M o n i t o r
3. S p e i c h e r v e r w a l t u n g
4. E i n / A u s g a b e
5. M a t h e m a t i k
6. B A S I C - I n t e r p r e t a t i o n

1. Initialisierung

Beim Einschalten des Rechners (Power on reset) holt sich der Mikroprozessor die Startadresse (Reset-Vektor) von den Speicherplätzen $FFFC/FFFD.
Dieser Reset-Vektor zeigt auf die Startroutine, die bei 64721<64790 beginnt.

Anmerkung: In der nun folgenden Untersuchung der Initialisierungsroutine beziehen sich fast alle Adressangaben auf die Zeropage. Die Kontrollregister der PIA's und VIA's, die auch auf definierte Werte gesetzt werden, werden hier nicht näher betrachtet. Für ein besseres Verständnis der Initialisierungsroutine sollte man sich die Speicheraufteilung und die Funktion der hierfür notwendigen Zeiger ansehen (siehe 3.1)

1.1 Ablauf der Initialisierung

In der Initialisierungsroutine wird zuerst der Stackpointer auf $FF gesetzt, d.h. er wird auf die oberste Stelle des Stapels ($0100-01FF) gerichtet, da der Stapel von oben nach unten gefüllt wird. Der Stapel kann bis zu 256 Bytes umfassen und hat eine LIFO-Struktur (last in,first out: das zuletzt eingegebene wird zuerst wieder ausgegeben).
Es wird nun die RESET-Routine aufgerufen, in der folgende wichtige Funktionen ausgeführt werden:

1. Alle Speicherzellen von 141-250 (Bedeutung siehe Anhang 27) werden gelöscht (= mit 0 beschrieben).
2. Der Interruptvektor für den Hardware-interrupt (ausgelöst durch den Bildwechsel-impuls) wird in (144,145) eingerichtet:
3. Der Bildschirm wird als Ausgabekanal festgelegt.
4. Die Blinkdauer des Kursors wird definiert.
5. Der CBM 3000 legt eine Tabelle der High-Bytes der Zeilenanfangsadressen (24 Zeilen) vom Bildschirm-RAM ($8000-$83FF) in der Zeropage ab. Die Tabelle der Low-Bytes der Zeilenanfangsadressen befindet sich im ROM.
   Der CBM 8000 legt an die gleichen Adressen ebenfalls Pointer für diverse Bildschirm-funktionen ab, jedoch mit unterschiedlichen Bedeutungen (s. Tabelle). Ausserdem besitzt der CBM 8000 einen CRT-Controller, der in der RESET-Routine ebenfalls programmiert wird.
6. Der Kursor wird in die linke, oberste Bildschirmecke gesetzt.
7. Der Bildschirm wird gelöscht, d.h. das ganze Bildschirm-RAM wird mit Blanks beschrieben.

Danach werden zwei weitere wichtige Interrupt-Vektoren angelegt:

1. Der NMI-Vektor in (148,149) wird auf die Adresse des BASIC-Warm-Start gerichtet.
2. Der Break-Vektor in (146,147) wird auf den TIM-Monitor gerichtet.
   Weiterhin wird in (1018,1019) ein Vektor eingerichtet, der auf die Monitor-Warteschleife zeigt.

Das Interrupt-Disable-Flag wird nun zurückgesetzt und der Diagnose-Eingang (USER-Port Kontakt 5) abgefragt.
Wenn dieser Eingang auf LOW liegt, erfolgt ein Aufruf des TIM-Monitors in (Call by entry). Steht dagegen eine 1 an diesem Eingang (üblicher Zustand), so wird die RESET-Routine aufgerufen.
In dieser Routine werden folgende wichtige Funktionen ausgeführt:

1. Der Maschinensprachekode für JMP ($4C) wird in 0 (für den BASIC-Befehl USR) und in 81 (für Funktionsaufrufe) abgelegt.
2. Als Ersatz-Sprungadresse für den USR-Befehl wird die Adresse 53539<50035 in (1,2) geschrieben, was bei Benutzung des USR-Befehles ohne vorherige Adressangabe den Aufruf der Fehlermeldung "ILLEGAL QUANTITY ERROR" bewirkt.
3. Die CHRGET-Routine (holt das nächste BASIC-Zeichen aus dem BASIC-Text) und die Zufallszahl werden aus dem ROM-Bereich in die Zeropage (112-139) kopiert. Dies hat den Vorteil, dass die CHRGET-Routine "angezapft" werden kann (was im ROM-Bereich unmöglich wäre), um z.B. neü BASIC-Befehle an die vorhandenen anzuhängen.
4. Als aktiver Eingabe-Kanal wird die Tastatur bestimmt, als Ausgabekanal der Bildschirm.
5. Die Länge des Disk-Basic-Befehlsstrings wird auf den Wert 0 gesetzt.
6. Es wird ein Speichertest durchgeführt:
   Der RAM-Bereich wird ab 1024 mit den zwei Bitmustern 01010101 und 10101010 getestet. Hierbei wird jede Speicherzelle mit den Bitmustern beschrieben und gleich darauf ein Vergleich mit diesen durchgeführt. Führt dieser Vergleich zu keiner Uebereinstimmung, so setzt das Betriebssystem die Adresse der ersten nicht vorhandenen RAM-Speicherzelle als Top of RAM fest. Diese RAM-Obergrenze wird in zwei Zeigern gespeichert, von denen der eine in (52,53) auf die RAM-Obergrenze und der andere in (48,49) auf den Beginn des letzten, verwendeten Strings zeigt.

7. In 1024 wird eine 0 abgespeichert (zum Aufruf des TIM-Monitors via Breakpoint) und der Programmanfangszeiger in (40,41) auf den Wert 1025 gesetzt.
8. Ausdruck der Meldung ### COMMODORE BASIC ### (bzw. BASIC 4.0)
   Nach Berechnung des freien Benutzerspeicherplatzes (=Obergrenze des RAM minus Beginn des BASIC-Programmes, d.h. (52/53)-(40/41)) erfolgt die Meldung :
   XXXXX BYTES FREE.
9. Aufruf der Subroutine NEW (50525<56548):

   a. In die ersten beiden Stellen des Programmspeichers (1025,1026), die die Verknüpfungsadresse von der ersten zur zweiten Programmzeile enthalten, wird eine 0 geschrieben: Der Rechner erkennt damit, dass kein BASIC-Programm abgelegt ist.
   b. Der Zeiger auf den Beginn der Variablentabelle in (42,43) wird auf 1027 gesetzt.
   c. Initiieren des Programmzeigers, d.h. der Programmzeiger, den die CHRGET-Routine benoetigt, wird auf den Wert des Programmanfangszeigers - 1 gesetzt.
   d. Der String-Anfangszeiger in (48,49) erhält den Wert des Zeigers auf die RAM-Obergrenze.
   e. Der Zeiger auf den Beginn der Arraytabelle in (44,45) und auf das Ende der gesamten Variablentabelle in (46,47) wird mit dem Variablenanfangszeiger in (42,43) gleichgesetzt.
   f. Der Zeiger auf das nächste DATA in (62,63) wird auf den Wert des Programmanfangzeigers -1 gesetzt.
   g. Schliessen aller Kanäle und Files.
   h. Disk-Statusvariable ds und ds$ löschen.

Damit ist die Initialisierung abgeschlossen und es erfolgt ein Sprung in die BASIC-Warm-Start Routine mit Ausdruck der READY-Meldung.
Das System befindet sich nun in einer Eingabewarteschleife.

Die Uebersicht auf der kommenden Seite enthält eine Aufstellung aller wichtigen Speicherplätze in der Zeropage ($0000-00FF), die während der Initialisierungroutine auf definierte Startwerte gesetzt werden.

Belegung der Zeropage während der Initialisierung

Adresse	Inhalt	Bedeutung
0	76	JMP-Befehl für USR
1,2	35/209	Aufruf von ILLEGAL QUANTITY ERROR als USR-Adresse
10	0	Flag für Dimensionierung bzw. für Funktionen
13	0	Flag für Output
14	0	Nummer des aktiven I/O-Kanals
19	22	Zeiger auf nächsten Descriptor
40,41	1/4	Programmanfangszeiger => 1025
42,43	3/4	Zeiger auf den Beginn der Variablen-tabelle => 1027
44,45	3/4	Zeiger auf den Beginn der Array-tabelle => 1027
46,47	3/4	Zeiger auf das Ende der Variablen-tabelle => 1027
48,49	0/128	Zeiger auf den Beginn des letzten Strings => 32768
52,53	0/128	Zeiger auf die höchste RAM-Adresse + 1 => 32768
59	0	High-Byte des Zeigers auf nächsten BASIC-Befehl = 0, d.h. CONT gesperrt
62,63	0/4	Zeiger auf nächstes DATA => 1024
80	3	Schrittweite für Garbage Collect
81	76	JMP-Befehl für Funktionsaufruf
112-135		Kopie der CHRGET-Routine aus ROM
136-139		Zufallszahl aus ROM
144,145	46/230	IRQ-Vektor => 58926 < 58453 (Tastaturabfrage)
146,147	23/253	Break-Vektor => 64791 < 54392 (=Einsprung in TIM-Monitor)
148,149	137/195	NMI-Vektor => 50057 < 46079 (BASIC-Warm-Start)
159	0	Flag für RVS-Darstellung => RVS = AUS
174	0	Anzahl der offenen Files = 0
175	0	Eingabekanal = Tastatur
176	3	Ausgabekanal = Bildschirm
213	39 23	Flag für einfache Zeile (CBM 8000: Spaltenende für Bildsch.fenster)
224-230	128	Tabelle der
231-236	129	High - Bytes der
237-243	130	Zeilenadressen im
244-248	131	Bildschirm-RAM (32768 bis 33767, bzw. bis 35767)

Beim CBM 8000 zusätzlich

224	0	Zeilenanfang des Bildschirmfensters
225	79	" ende " "
226	0	Spaltenanfang " "
227	9	Maximale Anzahl der Zeichen im Tastaturpuffer

A. Kaltstart	START	64721 < 64790

Voraussetzungen

Keine

Ablauf der Routine

Die Routine bewirkt dasselbe wie das Einschalten des Rechners (siehe Initialisierungsablauf).
Einen Sprung in diese Routine kann man dazu benutzen, um z.B. einen unautorisierten Eingriff in ein zu schützendes Programm mit einem Total-RESET zu beantworten, oder aber den Rechner in den Einschaltzustand zu versetzen, ohne die Peripheriegeräte zu beeinflußen.

B. RESET I	(nur beim CBM 3000 separat:)	57822=$E1DE

Voraussetzungen

Keine

Ablauf der Routine

Der Hardware-Interruptvektor zur Tastaturabfrage wird eingerichtet, die Kontrollregister in den PIA's und VIA's werden initialisiert und der Bildschirm gelöscht.

C. RESET II	INIT	57622 < 54198

Voraussetzungen

Keine

Ablauf der Routine

Die Routine kopiert die CHRGET-Routine in die Zeropage, führt den Speichertest durch und setzt alle Zeiger für die Speicherverwaltung auf ihre Anfangswerte.
Nach Ausgabe der Kopfzeile und der Anzahl der freien Bytes erfolgt ein Sprung in die BASIC-Warmstart-Routine, die sich mit '.READY' meldet.
Beim Aufruf der RESET II-Routine ist zu beachten, dass durch den Speichertest alle Speicherinhalte von 1024 bis zur Obergrenze des verfügbaren RAM zerstört werden.

D. BASIC-WARM-START	READY	50057 < 46079

Voraussetzungen

Das System muss bereits einmal vollständig initialisiert worden sein.

Ablauf der Routine

Nach Ausgabe der READY-Meldung befindet sich das System in einer Warteschleife, in der jedes eingegebene Zeichen bis zum ersten CR in den INPUT-Puffer (512-592) zwischengespeichert wird. Danach wird bei einem Direktkommando nach Umwandlung des Eingabetextes in den entsprechenden BASIC-Kode zur Interpreterschleife verzweigt. War es eine BASIC-Zeile, so wird diese nach ihrer Codierung im Programmspeicher abgelegt.

Zwei Hinweise, die sich aus der genauen Untersuchung der Initialisierungsroutine ergeben, sollen hier noch genannt werden:

1. Wenn sich der Rechner nach der Initialisierungsphase mit einem geringeren Speicherplatz meldet, als ihn das Systems besitzt, so zeigt das nur an, dass mindestens **ein RAM defekt ist.**
   Da das System bei einer Nichtübereinstimmung des Vergleiches der Bitmuster die Testroutine sofort abbricht, kann nicht darauf geschlossen werden, dass in einem solchen Fall alle RAM's ab dieser bestimmten Adresse (1024+Anzahl der freien Bytes) defekt sind.

2. Besonders bei der Assemblerprogrammierung kann es vorkommen, dass sich das System "aufhängt".
   Um die Daten im RAM zu retten, kann man den DIAGNOSTIC Eingang auf Masse ziehen (USER-Port Kontakt 5 mit 12 verbinden) und danach einen TOTAL-Reset des Rechners ohne Ausschalten durchführen.(siehe 2.6)
   Damit erreicht man, dass man in den TIM-Monitor kommt, ohne dass die Daten durch die Speichertestroutine überschrieben werden.

2. Der TIM-Monitor

Zum Betriebssystem des CBM-Rechners gehört auch der sogenannte TIM (terminal interface monitor), der im folgenden erläutert werden soll.

Dieser Monitor ermöglicht die Kommunikation mit dem Rechner im Hexadezimalsystem auf der untersten Ebene, der Maschinensprache.
Er erlaubt hier das Ansehen und Aendern von Speicherstellen und Registerinhalten und damit auch von Maschinenprogrammen oder (sehr umständlich!) einem Basicprogramm.

Sehr wichtig ist auch die in ihm implementierte Möglichkeit, Maschinenprogramme direkt auf Kassette abspeichern zu koennen.

Allerdings werden die Grenzen dieses Monitors dem engagierten Programmierer bald zu eng werden, da das Editieren von Programmen in Maschinensprache damit sehr mühsam ist!
Das Hexadezimalsystem ist sehr gewöhnungsbedürftig und die fertigen Programme sind unübersichtlich sowie fehlerfreudig bei der Eingabe.

Es empfiehlt sich daher bei intensiver Maschinenprogrammierung, auf einen der käuflichen Assembler und Disassembler zurückzugreifen und so durch komfortable Adressierungsmöglichkeiten und verwechslungssichere Codes das Rechnen im Hexadezimalsystem auf ein Minimum zu beschränken und den Programmieraufwand auf einen Bruchteil herabzusetzen.

Diese Assembler und Disassembler sind Hilfsprogramme, die es ermöglichen, Maschinenprogramme direkt im mnemonischen Code einzugeben und - je nach Komfort - auch symbolische Adressierungen zulassen.
(Die Maschinenprogrammlistings in dieser Beschreibung sind mit Hilfe eines solchen Programmes entstanden).

Auf dem Markt sind die verschiedensten Versionen erhältlich, die gewisse Vor-und Nachteile aufweisen:

Die Versionen, die mit Kassettenstationen arbeiten, sind immer bedeutend langsamer als die Floppy-Disk-orientierten Programme.

Die Programme, die in Maschinensprache geschrieben sind, benötigen im allgemeinen weniger Speicherplatz und sind zudem wesentlich schneller als die Programme in Basic.

Die Programme in Maschinensprache setzen meist fundierte Kenntnisse über den vom CBM-Betriebssystem verwalteten Speicherbereich voraus.

Mit dem Komfort steigt erfahrungsgemäss auch der Preis eines solchen Hilfsprogrammes.

(Bem.: die Verfasser geben gerne Auskunft über Bezugsquellen, sowie Hilfestellungen bei der Auswahl zwischen den verschiedenen Versionen).
Im einzelnen können diese Programme in diesem Rahmen allerdings nicht weiter erläutert werden.

Im folgenden sollen vielmehr die Funktionen des installierten CBM-Monitors erläutert werden.

2.1 Die Arbeit mit dem TIM

Um in den TIM zu gelangen, bieten sich zwei Möglichkeiten an:

a. Durch den Sprung auf ein BREAK-Kommando ($00), das z.B. immer auf den Speicherstellen 4 und 1024 steht (es sei denn, es wurde manipuliert), also über die Basic-Befehle SYS 4 oder SYS 1024.
Diesen Einsprung nennt man auch den "BREAK-Entry".
Der Monitor meldet sich dann mit:

```
B*
    PC  IRQ  SR AC XR YR SP
.; **** **** ** ** ** ** **
```

Hierbei bedeuten die Angaben unter den Abkürzungen der Register- bzw. Zeigernamen folgendes:

PC: (Program Counter) Die nächste, auszuführende Adresse wäre die Adresse gewesen, die hier angegeben ist.
Dies ist im Falle des Einsprunges mit
SYS 4 die Adresse $0005 und bei
SYS1024 die Adresse $0401.

IRQ: (Interrupt Request) Diese Adresse zeigt an, wohin der IRQ-Vektor deutet. Dies ist im Normalfall die Adresse $E62E > e455, bei der die Routine des Betriebssystems zur Uhr-Steuerung und Tastatur-Abfrage beginnt, welche jede 1/60 Sekunde durch das Interruptsignal vom Bildwechselimpuls aufgerufen wird.

SR: (Status Register) Dieses Byte gibt Informationen über die verschiedenen Flags ab, die im Programm gesetzt und gelöscht werden und ist folgendermassen aufgeschlüsselt:

```
N V * B D I Z C
```

Wobei bedeuten

- N - Ergebnis negativ
- V - Overflow
- * - Nicht verwendet
- B - Break-Kommando
  (wird vom Prozessor selbst gesetzt und kann vom Programmierer nicht beeinflusst werden)
- D - Flag für Dezimalbetrieb
- I - Flag für Interrupt-Zulassung
- Z - Ergebnis ist Null
- C - Carry-Flag

AC: Inhalt des Akkumulators

XR: Inhalt des X-Registers

YR: Inhalt des Y-Registers

SP: (Stack Pointer) Dieser Zeiger gibt an, an welcher Stelle des Stacks das nächste Byte abgelegt würde. Näheres über die Struktur des Stacks siehe (2.7).

b. Der Einsprung über SYS 64785 (< 54386) unterscheidet sich vom Fall a. zunächst einmal optisch dadurch, dass der Monitor sich meldet mit ("Call Entry"):

```
        CBM 3000:                          CBM 8000:
C*
    PC  IRQ  SR AC XR YR SP            PC  IRQ
.; FD12 E62E ** ** ** ** **           D473 E455
.
```

Der nächste Unterschied liegt darin, dass der C*-Einsprung den Inhalt des Stackpointers nicht verändert - im Gegensatz zum B*-Einsprung, der ja den SP um 2 Stellen erniedrigt. Ausserdem holt er alle Register und Zeiger vom Stack und legt sie - was Routine a. unterlässt- im BASIC-Input-Puffer ab:

```
PC  => ($0200/0201) = (HB/LB)
IRQ => ($0207/0208) = (HB/LB)
SR  => ($0202)
AC  => ($0203)
XR  => ($0204)
YR  => ($0205)
SP  => ($0206)
```

Hier treffen sich die beiden Routinen und laufen in die Monitor-Warteschleife, in der sie eines der Kommandos erwarten.

2.2. Die Monitor-Warteschleife: STRT 64854 < 54458

Der Monitor erwartet nach der Ausgabe von "." die Eingabe eines Kommandos. Dabei ignoriert er alle folgenden "." und SPACES, die man auch willkürlich eingeben könnte.
Andere Zeichen vergleicht er mit seiner Befehlstabelle, die aus den folgenden 8 Befehlen besteht und in ($FDE0-FDE7 < $D544-D54B) abgelegt ist:

: ; R M G X L S

Diesen Vergleich führt er aus, indem er den Inhalt des AC (=eingegebenes Kommando) mit der genannten Tabelle vergleicht.

Entspricht der Code im AC einem gültigen Kommando, so erhält das XR dessen Stellennummer in der Tabelle.

Der Monitor lädt nun XR-indiziert den AC mit dem HB der Einsprungadresse der zugehörigen Routine (HB-Tabelle in $FDE8-FDEF < $D54C-D553) und schiebt den AC auf den Stack. Das gleiche macht er mit dem LB (LB-Tab. in $FDF0-FDF7 < D554-D55B) der Einsprungadresse.

Der danach folgende RTS-Befehl holt sich die "Rücksprungadresse" vom Stack und springt somit die gewünschte Routine an. Diesen kleinen Programmiertrick sollte man sich merken: Man kann damit quasi die ON-GOTO-Sprungverteilung im BASIC imitieren.

Näheres über das Handling des Stack siehe auch (2.7).

Hat der Monitor unter den eingegebenen Zeichen kein gültiges Kommando gefunden, so springt er auf die Adresse, die im Zeiger ($03FA/03FB) abgelegt ist.
Im Normalfall ist dies die Adresse $E7F7 < D7A4 (=ERROR-Routine des Monitors / Ausgabe von "?" / Sprung in die Monitor-Warteschleife). Diese kann allerdings auch auf eine eigene Routine zeigen, die z.B. den Monitor-Befehlssatz erweitern könnte!

2.3 Befehle des Monitors

Die Routinen des Monitors können auch als Subroutinen durch den direkten Einsprung verwendet werden:

Der ":"-Befehl: Speicherstellen ändern ALTM 65209 < 54813

Wenn der Monitor das Zeichen ":" am Zeilenanfang findet, so springt er in diese Routine.
Er überliest dann bis zu 2 Zeichen des Bildschirms direkt hinter dem ":" und erwartet danach eine 4-stellige Hexadresse, die er nach ($FB,FC) abspeichert (JSR RDOA= $E7A7 < D754).
Hat er diese gefunden, so liest er jeweils im Anschluss daran ein Byte und ändert es gegebenenfalls durch Abspeichern an seine Adresse im Speicher (JSR BYTE= $FDA7 < D50B).
Dies macht er solange, bis die Bildschirmzeile zu Ende ist, also 8 mal, um dann wieder in die Warteschleife (STRT= $FD56 < D4BA) zu springen.

Der ";"-Befehl: Registerinhalte ändern ALTR 65175 < 54779

Dieser Befehl dient der Änderung sowohl der Register PC, SP, XR, YR und AC als auch des IRQ-Vektors und des Satusregisters SR.
Findet der Monitor am Zeilenanfang ein ";", so erwartet er, nachdem er bis zu 2 Spaces ignoriert hat, die hexadezimale Adresse des Programmzählers (PC) und ändert diesen auf den angegebenen Wert.
Nach Ueberlesen von weiteren 2 Spaces liest er den Interruptvektor (IRQ) und ändert ihn gegebenenfalls.
Die Register liest und ändert er wenn nötig mit der gleichen Routine, wie sie bei der Änderung der Speicherstellen beschrieben ist.
Dabei werden diese nicht direkt verändert, sondern vielmehr ihr Zwischenspeicher im RAM. Erst durch den nächsten ".G"-Befehl werden sie aktualisiert (d.h. mit den Werten aus den Zwischenspeichern geladen).
Danach Rücksprung zur Monitor-Warteschleife.

ACHTUNG: Auch der IRQ-Vektor wird durch das Editieren nicht sofort geändert, sondern sein neuer Wert wird zunächst in den BASIC INPUT BUFFER abgelegt und erst durch den nächsten ".G"-Befehl aktualisiert.

Der "R"-Befehl: Registerinhalte zeigen DSPLYR 65059 < 54663

Wenn der Monitor am Zeilenanfang ein "R" findet, so druckt er nach einem Zeilenvorschub zunächst die Kopfleiste für die Registerausgabe aus.
Dazu holt er sich die Zeichen XR-indiziert aus der Tabelle $FDF8-FE18 in den AC und gibt diese (=29 incl. Blanks) auf den Bildschirm aus.
Sodann holt er sich die Zeiger aus den Speicherzellen $0200/0201 (=PC) und $0207/0208 (=IRQ) und druckt sie hexadezimal auf den Bildschirm (JSR WROB= $E775 < D722).
Die Register werden durch eine Routine ausgegeben, die noch weiter unten unter dem "M"-Befehl beschrieben wird.

Der "M"-Befehl: Speicherinhalte zeigen DSPLYM 65112 < 54716

Wenn der Monitor das Zeichen "M" am Zeilenanfang der aktuellen Zeile findet, so überliest er das darauffolgende Zeichen (beliebig, üblicherweise ein Blank) und erwartet sodann eine 4-stellige Hexadezimalzahl, die ihm die Startadresse des Memory-Dumps angibt.
Nach Ueberlesen des nächsten Zeichens (=Trennungszeichen, beliebig, meist wird ein Blank verwendet) holt er sich die nachfolgende 4-stellige Hexadezimalzahl als Endadresse.
Die Anfangs- und Endadresse befinden sich nun in den Speicherzellen ($FB,FC) und ($FD,FE).

Nach Ueberprüfung der Eingaben druckt der Monitor Zeile für Zeile die Anfangsadressen auf den Bildschirm aus, gefolgt von den Angaben über den Inhalt der 8 folgenden Speicherstellen (diese Ausgaberoutine der Speicherinhalte wird auch von der "R"-Routine benutzt).
Dies geschieht solange, bis die letzte ausgedruckte Speicherzelle einer Zeile eine höhere Adresse besitzt als die eingegebene Endadresse. Ist dies der Fall, so springt der Monitor in seine Warteschleife und erwartet den nächsten Befehl.

Der "G"-Befehl: Masch.-Programm aufrufen	MGO	65231 < 54835

Nach Auffinden des "G"-Kommandos überliest der Monitor bis zu 2 Blanks und liest dann die Einsprungadresse des Maschinenprogrammes als 4-stellige Hexadezimalzahl, mit der er den PC lädt.
Falls er nur Spaces nach dem "G"-Befehl findet, so startet er das Programm an der Stelle, auf welcher der PC steht.

> Bei Benutzung dieses Befehles sind die Hinweise unter (1.5 Eigenarten des CBM-Monitors) zu beachten!

Der "X"-Befehl: Basic-Warmstart	EXIT	65287 < 54891

Findet der Monitor in der Zeile als erstes Zeichen (nach dem ".") das Kommando "X", so lädt er den Stapelzeiger mit dem Wert der Speicherstelle $0206 und springt dann in den Basic-Warmstart (READY= $C389 < B3FF).
Dort druckt er zunächst "READY." aus und befindet sich jetzt in der Eingabe-Warteschleife.

Die Befehle "L" und "S"	LD	65297 < 54901

Laden und Abspeichern von Maschinenprogrammen.

Wenn der Monitor eines dieser Zeichen findet, so erwartet er nach einem Blank ein Hochkomma.
Findet er stattdessen ein CR, so führt er sofort die Routine a. (=LOAD, siehe unten) durch.
Ansonsten wird jedes darauffolgende Zeichen bis zum nächsten Hochkomma als Teil des Dateinamens in den Speicherbereich ab ($DA,DB) geschrieben. Der Zähler für die Länge des Dateinamens ($D1) wird incrementiert.
Wird der Name länger als 16 Zeichen, so erfolgt Fehlermeldung.

Bis hierher sind die Routinen "S" und "L" gemeinsam gelaufen und trennen sich nun:

a. Routine für "L"

Der Monitor springt in den Monitor-Eingang der LOAD-Routine (LD15= $F322 < F356, siehe 4.4.7.2).
Nach der Rückkehr fragt er das Bit 4 des I/O-Statusbytes ab.
Ist dieses gesetzt, so erfolgt die Meldung "LOAD ERROR".
Ist es dagegen gelöscht, so springt der Monitor in seine Warteschleife.

b. Routine für "S"

Der Monitor holt sich nun das nächste Zeichen hinter dem 2.Hochkomma und erwartet ein Komma.
Danach liest er als nächste Zeichen die Gerätenummer für die Ausgabe, wiederum ein Komma, die Anfangsadresse und nach einem weiteren Komma die Endadresse des Speicherbereiches, der auf das Peripheriegerät ausgegeben werden soll.
Dabei muss die Endadresse um ein Byte höher angegeben werden als die Adresse des letzten zu schreibenden Bytes.
Als mögliche Peripheriegeräte schliesst er Bildschirm und Tastatur aus.
Die Anfangs-und Endadressen werden in ($FD,FE) bzw. ($C9,CA) abgelegt.
Die Anfangsadresse befindet sich ausserdem noch in ($FB,FC) =TAPE-START-Adresse.

Bei "L." und "S." ist die Rubrik "Eigenarten des CBM-Monitors" (2.5) unbedingt zu beachten!

Die Routinen für das Laden und Abspeichern werden in (4.4.7) noch näher beschrieben.

Wenn der Monitor in den oben angegebenen Routinen auf ein Zeichen stösst, das er nicht interpretieren kann, so springt er in diese Routine:
Er druckt ein "?" hinter dem fragwürdigen Zeichen aus und springt dann in seine Warteschleife, um den nächsten Befehl zu holen.

2.5 Eigenarten des CBM-Monitors

a. Die vom Monitor angezeigten Registerinhalte werden erst mit dem darauffolgenden "G"-Befehl eingesetzt. Dadurch kann speziell der gezeigte IRQ-Vektor nicht einfach durch Editieren geändert werden, da der neue Inhalt ebenfalls erst mit "G" wirksam wird.
Wenn der Stackpointer in der Nähe von 0 ist, so ist Editieren ohne darauffolgendes "G" ebenfalls sinnlos, da der angezeigte Stackpointer nur ein Byte aus dem BASIC INPUT BUFFER ist und erst durch "G" in den Stackpointer der CPU 6502 eingesetzt wird.

b. Der Monitor legt seine Dateinamen, die zu "L." und "S." gehören genau da ab, wo er eigentlich den IRQ-Vektor für sich bereithält.

   Das bedeutet, dass nach LOAD oder SAVE auf keinen Fall mit dem "G"-Befehl weitergemacht werden darf, da sonst die ersten beiden Zeichen des Dateinamens als neuer IRQ-Vektor interpretiert werden.

   Der IRQ-Vektor muss also jedesmal manuell mit dem ";"-Befehl (siehe 2.3) geändert werden.

2.6 Nützliche Hilfen für den Maschinenprogrammierer

Bei der Programmierung des CBM in Maschinensprache, wie auch in Basic (bei Verwendung von PEEK's und POKE's) passiert es leicht, dass der Rechner sich in einen Zustand begibt, in dem er weder durch die STOP-Taste noch durch eine andere Taste kontrollierbar ist und auf dem Bildschirm entweder wirre Zeichen oder gar nichts mehr ausdruckt - der Rechner "steigt aus"!
Bei den serienmässigen Commodore-Rechnern ist dann ein ordnungsgemässes Arbeiten nur durch Aus- und Wiedereinschalten der Stromversorgung (Netzschalter) zu erreichen.
Die dabei auftretenden hohen Spannungsspitzen belasten das Videoteil sehr, weshalb viele CBM - Benutzer ihren Rechner mit der schonenden RESET - Taste ausgerüstet haben, bei deren Betätigung lediglich ein Sprung in den Kaltstart (INIT= $E116 < D3B6) erzwungen wird.
Dieser Sprung bewirkt allerdings auch den Verlust von Programmen, welche ab 1024 im Speicher stehen.

Dies kann man durch Einbau von drei zusätzlichen Tasten verhindern:

RES-Taste

Diese Taste erzwingt nach Abarbeiten des momentanen Maschinenbefehls einen Sprung in den Kaltstart INIT ($E116 < D3B6) =TOTAL RESET. Das entspricht dem Aus- und Wiedereinschalten der Stromversorgung; gelöscht wird allerdings nur der Speicherbereich ab der Adresse 1024.
Diese Taste ist vielen CBM-Benutzern schon bekannt.

NMI-Taste

Diese Taste bewirkt nach Beendigung des aktuellen Befehls eines laufenden Maschinenprogrammes einen Sprung in die BASIC-Warteschleife, gleichgültig, in welchem Arbeitsgang sich der Rechner befindet.
Die Adresse für diesen Sprung steht im Zeiger ($94,95) und kann auf die Anfangsadresse einer eigenen Routine verbogen werden.

IRQ-Taste

Eine Betätigung dieser Taste erzwingt den sofortigen Einsprung des Rechners in die Routine, deren Anfangsadresse im Zeiger ($90,91) steht.
Im Normalfall ist dies die Routine zur Uhr-Steuerung und Tastatur-Abfrage J58453 ($E62E < E455), welche er immer wieder abarbeitet, solange die IRQ-Taste gedrückt ist, um danach zu der Routine zurückzukehren, die er bei Betätigung der Taste verlassen hat.
ACHTUNG! Die Uhr wird dabei vorgestellt!

Durch Verbiegen des IRQ-Zeigers ($90,91) auf den Anfang einer eigenen Routine kann erreicht werden, dass der Rechner sofort bei Betätigen der IRQ-Taste in diese Routine springt.

2.6.1 Einbau der Tasten

Die Tasten können einfach durch Verbindung der entsprechenden Pins des 50-poligen Expansionboards mit Masse realisiert werden.

Die Anschlüsse sind wie folgt zu finden:
Schaut man von rechts auf die CBM-Platine, so sieht man das Expansionboard mit 2 parallelen Reihen von je 2*25 Anschlusspins.
Die vordere Reihe (A) ist mit der Gerätemasse verbunden.
Beide Reihen teilen sich auf in die beiden Steckverbindungen J4 (links) und J9 (rechts) und sind jeweils von links her nummeriert.

Die gesuchten Pins liegen nun wie folgt:

RES = Reihe B, Stecker J4, Pin Nr.22
NMI = Reihe B, Stecker J4, Pin Nr.24
IRQ = Reihe B, Stecker J9, Pin Nr.19
GND = Reihe B, Stecker J4, Pin Nr.25

Das Auffinden und Anschliessen dieser Pins dürfte auch dem Laien keine Schwierigkeiten bereiten.
Es ist jedoch unbedingt darauf zu achten, dass der CBM währenddessen ausgeschaltet ist!

2.6.2 Anwendungen der Tasten

a. RESET-Taste

Der Rechner führt die Initialisierung durch (siehe 1.1) und löscht dabei den Speicherbereich ab 1024.

b. IRQ-Taste

Bewirkt Sprung nach ($90,91) - normalerweise zur Adresse
J58453 ($E62E < E455) = Uhr-Steuerung & Tastaturabfrage. Insbesondere kann aufgrund dieser Funktion mit der Taste folgende Wirkung erreicht werden:
Soll ein BASIC-Programm gelistet und dabei angehalten werden, so wird normalerweise die STOP-Taste betätigt.
Die LIST-Funktion wird dabei beendet und muss erneut gestartet werden. Auch die RVS-Taste verlangsamt nur den Ablauf des Listings.

Das Gleiche gilt für ein laufendes Programm: Es kann zwar verlangsamt oder ganz abgebrochen, nicht aber angehalten werden, um danach weiterlaufen zu können.

Dies erlaubt aber nun die IRQ-Taste:

Wird sie gedrückt, so werden Programm oder Listing angehalten und nach Loslassen weiter bearbeitet, als wäre gar nichts geschehen.

Zu beachten ist, dass dabei die interne Uhr des Rechners weitergestellt wird !

Beim CBM 8000 ist diese Funktion bereits softwaremässig implementiert:
Durch Betätigen der Taste ':' wird die Ausgabe angehalten und mit '3', '6' oder '9' fortgesetzt.

c. NMI-Taste

Mit dieser Taste wird ein Abbruch eines laufenden Programmes ohne Datenverlust erreicht.
Das Programm kann danach erneut gestartet oder gelistet werden.

Manchmal wird es vorkommen, daß der Einsatz dieser Taste keine Wirkung zeigt.
Dies ist dann der Fall, wenn der NMI-Vektor in der Zeropage (=indirekter Sprungvektor) verbogen ist.
Dies kann durch ein gezieltes Setzen im Programm geschehen sein, wie auch durch einen Fehler, der den Rechner "aussteigen" ließ: er läuft unkontrolliert weiter.
In diesem Fall helfen nur noch zwei Mittel:

1) Aus- und Wiedereinschalten der Netzspannung (bzw. RESET-Taste) oder

2) Bedienung der 3 neuen Tasten in folgender Reihenfolge:

a. Zuerst die RESET-Taste drücken und festhalten.

b. Danach die IRQ-Taste drücken und festhalten, die RES-Taste wieder loslassen.

c. Zuletzt die NMI-Taste zusätzlich betätigen und die IRQ-Taste freigeben.

Nach Loslassen der NMI-Taste befindet sich der Rechner wieder im READY-Modus, ohne Daten- oder Programmverlust. Allerdings kann ein "durchgegangenes" Programm alle möglichen Zeiger in der Zeropage verstellt haben, so daß dann beispielsweise keine BASIC-Befehle mehr erkannt werden. Dann hilft allerdings leider nur noch das Ausschalten.

Mit diesen kleinen Zusätzen ist nun die Voraussetzung geschaffen, auch in Maschinensprache ohne größere Zeitverluste beim Programmtest zu arbeiten, da nun da wiederholte Ausschalten des Rechners und lästige Laden des Programmes nach fehlerhaften Routinen entfällt.

2.7 Der STACK des CBM

Die Ausführungen der Möglichkeiten, welche der Stack bietet, fallen gegenüber den Behandlungen des übrigen Betriebssystems quantitativ aus dem Rahmen.

Nachdem jedoch gerade das MICROSOFT-BASIC diese Möglichkeit der effektiven Programmierung oft benutzt, wird eine detaillierte Beschreibung des Stacks als besonders interessant erachtet.
Ausserdem wird gerade der Stack in der uns bekannten 6502-Literatur vernachlässigt.

Diese Abhandlung soll nun dazu helfen, durch gezielte Verwendung des Stacks umständliche Programmiermethoden zu umgehen, bzw. zeitkritische Programme zu beschleunigen:

Der Stack ist ein Speicherbereich ($0100 bis 01FF), auf den das Betriebssystem schnell und einfach zugreifen kann.
Beim CBM ist er auf den Bereich zwischen $0140 und $01FA begrenzt.

Die Verwendung des Stack bietet sich vor allem dann an, wenn Daten kurzzeitig abgespeichert werden sollen. Diese können dann auf den Stack gegeben und später wieder abgerufen werden.
Wohin diese Daten gelangen, braucht für den Programmierer nicht weiter von Interesse sein, da der Mikroprozessor diese durch ein spezielles Register (= Stackpointer) lokalisiert:

> Der Mikroprozessor "schiebt" Informationen auf den Stack und holt sie zurück, ohne dabei zusätzliche Hinweise zu benötigen.
> Dieses "Schieben" kann man auch in eigenen Programmen verwenden. Allerdings muss man dabei stets darauf bedacht sein, den Stack immer wieder genau so verlassen, wie man ihn vorgefunden hat!
> Werden also Daten auf den Stack abgelegt, so müssen diese auf jeden Fall zurückgeholt werden, oder es gibt bei der Fortsetzung des Programmes Schwierigkeiten!

Nach der Art seiner Organisation trägt der Stack (=Stapel) seine Bezeichnung.

> Man könnte ihn mit einem Stapel Teller verglei-
> chen:
>
> > Der Teller, der als letzter auf den Stapel kommt, verlässt diesen auch zuerst.
> > Der Stack wird daher auch LIFO (last in - first out) -Speicher genannt.

Uebliche Verwendung für den Stack

a. Der Inhalt des AC, der für kurze Zeit zwischengespeichert werden soll, kann durch den Befehl PHA (Push AC=$48) auf den Stack geschoben werden.
Dadurch ist der AC frei für anderweitige Verwendung und kann nach Beendigung der Operationen durch den Befehl PLA (Pull AC = $68) wieder mit dem Originalwert aus dem Stack geladen werden.

b. Eine andere Möglichkeit zur Nutzung des Stacks:

Sind Ergebnisse eines Vergleichs oder ähnlicher Operationen zwar wichtig, können aber im Moment aus irgendeinem Grund noch nicht bearbeitet werden, so kann das Statuswort mit all seinen verschiedenen Flags wie Carry, Overflow usw. mit Hilfe von PHP (=Push Processor Status =$08) auf den Stack gebracht werden.
Man kann nun mit den Registern arbeiten, ohne sich um diese Flags zu kümmern, da diese erst dann wieder gültig sind, sobald man den Befehl PLP (=Pull Processor Status =$28) gibt. Danach kann z.B. mit den Verzweigungsbefehlen fortgefahren werden, welche die zuvor aufgestellten Vergleichsergebnisse prüfen.

c. Wird eine Subroutine mit dem JSR-Befehl aufgerufen, so tritt automatisch der Stack in Aktion:
Der Programmzählerinhalt plus 2 wird auf den Stack abgelegt. Dieser Wert entspricht der dem JSR-Befehl folgenden Adresse minus eins (=Rücksprungadresse -1).
Nach Beendigung des Unterprogramms wird (durch den RTS-Befehl) diese Adresse in den Programmzähler geladen und inkrementiert, wodurch der Programmablauf mit dem Befehl hinter JSR fortgesetzt wird.

Beispiel:

Ein Programm beginne bei der Adresse $1234 mit 'JSR $4455'
Dieser erste Befehl bewirkt, dass die Adresse $1236 auf den Stack geschoben wird.
Dieser Wert entspricht nicht der Rückkehradresse, da ja bei der Rückkehr auf $1237 gesprungen werden soll (Der JSR-Befehl benötigt 3 Bytes).
Der RTS-Befehl wird jedoch alles korrekt ausführen (Inkrementierung der Rücksprungadresse).

Betrachtet man das Ganze genauer, so benötigt die Adresse $1236 zwei Plätze im Stack. Der höherwertige Teil (=$12) kommt zuerst auf den Stack, gefolgt vom niederwertigen Teil (=$36).

d. Erhält der Mikroprozessor einen Interrupt - und das geschieht beim CBM 60 mal in der Sekunde - so wird zunächst der aktuelle Maschinensprache-Befehl beendet.
Die Adresse des nächsten Befehles und schliesslich auch das Prozessor-Status-Wort werden sodann auf den Stack geschoben.
Erst dann verarbeitet der Prozessor den Interrupt, indem er zu der Adresse springt, auf die der IRQ-Vektor ($FFFE,$FFFF) bzw. der NMI-Vektor ($FFFA,FFFB) zeigt, um ab dort die Befehle auszuführen.

Sobald er die Interrupt-Routine beendet hat, erwartet er einen RTI-Befehl (Return from Interrupt), bei dem er das ursprüngliche Status-Wort und die Fortsetzungsadresse zurückholt.

Das ursprüngliche Programm fährt genau an der Stelle fort, an der es beim Interrupt verlassen wurde.

Es wurden diesmal also 3 Plätze des Stack in folgender Reihenfolge belegt:
zwei für die Rücksprungadresse und einer für das Status-Wort.

a. HB-Adresse
b. LB-Adresse
c. Status-Wort

Dies ähnelt dem JSR-Befehl, gefolgt von einem PHP.
Der wesentliche Unterschied liegt darin, dass hier die genaue Rücksprungadresse abgelegt wird, gegenüber der Rücksprungadresse minus 1 beim JSR-Befehl.

Beispiel: Der Prozessor führe bei Adresse $1234 einen 3-Byte-Befehl aus und erhält eine Interruptanforderung. Daraufhin schiebt er die Adresse $1237 auf den Stack, gefolgt vom Status-Wort.

Später, wenn RTI ausgeführt wird, holt er das Status-Wort zurück und setzt das Programm ab Adresse $1237 fort.

e. Schliesslich verursacht auch der BRK-Befehl ($00) einen Software-Interrupt mit folgender Wirkung: Der Wert der Programmzählers+2 und der SP werden auf den Stack gerettet und der Prozessor springt zu der in ($FFFE,FFFF) angegebenen Adresse.
Die Adresse, die im Stack abgelegt wird, zeigt also auf das zweite Byte hinter dem Break-Befehl.
Dies ist ungewöhnlich, da der BRK-Befehl ja nur ein Byte lang ist, und kann manchmal zu Schwierigkeiten führen, wenn man den geretteten PC-Wert nicht passend justiert.
Der Grund liegt darin, dass man oft einen BRK-Befehl bei der Fehlersuche an Stelle eines anderen Befehls einsetzt. Die meisten 6502-Befehle umfassen aber zwei Bytes, so dass man sich hier in der Mehrzahl der Fälle unnötige Korrekturarbeiten erspart, indem das Programm nach Rückkehr aus einer BRK-Unterbrechung (mit RTI) wieder genau am Anfang des nächsten Befehls steht.

Die folgende Tabelle stellt eine Zusammenfassung der Befehle dar, die mit dem Stack operieren:

Anzahl der benötigten Plätze	Speicherbefehl	Rückholbefehl
1	PHA	PLA
1	PHP	PLA
2	JSR	RTS
3	INTERRUPT	RTI
3	BRK	---

Mit dieser Tabelle ist nun die Beschreibung der konventionellen Methoden zur Arbeit mit dem Stack erschöpft.

Im folgenden soll darauf eingegangen werden, wie man durch gezieltes Verlassen dieser Regeln äußerst nützliche Effekte erreichen.

Verwendung des Stacks im Maschinenprogramm

Die beschriebenen Fakten ermöglichen interessante Manipulationen!

Bei der üblichen Programmierweise folgt einem Befehl zum Füllen des Stacks immer der zugehörige Befehl, um den Stack zurückzusetzen:
Wenn ein PHA verwendet wurde, folgt ihm beispielsweise ein PLA, um die Information zurückzuholen. Einem JSR folgt ein RTS usw.

Beim Verlassen dieser Reihenfolge eröffnen sich dem Programmierer völlig neue Wege:

a. Soll eine Subroutine geschrieben werden, so wird normalerweise die Kontrolle an die aufrufende Routine durch den RTS-Befehl zurückgegeben. Manchmal allerdings ist eine Rückkehr dorthin nicht erwünscht - beispielsweise aufgrund einer Fehlerbedingung - sodaß die aufgerufene Routine nicht weiterarbeiten könnte. Dies wird ermöglicht durch Beseitigung der Rücksprungadresse durch zwei PLA-Befehle. Somit ist eine Rückkehr ausgeschlossen. Der nächste auftretende RTS würde dann in die Routine springen, welche das Programm aufgerufen hat, in das nicht mehr zurückgekehrt werden soll. Es kann auch ohne RTS weiter gearbeitet werden, wenn gar kein Rücksprung erforderlich ist. Der Stack wurde auf alle Fälle ordnungsgemäß verlassen.

Beispiel: Das Hauptprogramm A ruft ein Unterprogramm B auf. Unterprogramm B ruft seinerseits Unterprogramm C mehrmals auf.
Vom Unterprogramm C soll aus irgendeinem Grund nicht in Unterprogramm B zurückgesprungen werden. Es soll also direkt in das Hauptprogramm A zurückgekehrt werden. Sobald Unterprogramm C aufgerufen wurde enthält der Stack 4 Werte: 2 für die Rücksprungadresse nach A und 2 für die Rücksprungadresse nach B.
Wird im Unterprogramm C nun die Sequenz PLA-PLA-RTS verwendet, so wird dabei die Rücksprungadresse nach B entfernt und direkt nach A zurückgesprungen.

b. Bekanntlich kann man mit dem JMP-Befehl auf jede gewünschte Adresse springen.

Es könnte allerdings auch einmal beabsichtigt werden, in Abhängigkeit von einer berechneten Variablen auf eine von mehreren Adressen zu springen.

Beispielweise soll ein Programm geschrieben werden, welches zu einer bestimmten Routine springt, wenn es das Zeichen "A" (Addition) findet, zu einer anderen bei Vorfinden des Zeichens "S" (Subtraktion) und zu einer dritten beim Zeichen "C" (Löschen) usw.

Dies könnte durch eine Folge von Vergleichs-, Verzweigungs-, und Sprungbefehlen erreicht werden. Wird diese Folge jedoch zu lang, so verliert die ganze Sache an Uebersichtlichkeit und erweist sich als unvorteilhaft.

Die gleiche Wirkung kann bei sinnvoller Nutzung des Stacks durch einen berechneten Sprung erzielt werden, der zudem viele neue Möglichkeiten der Verzweigung bietet:

Dazu wird im Prinzip eine Adresse geschaffen und mit PHA auf den Stack geschoben, gefolgt von RTS. Das Ganze erscheint auf den ersten Blick verwirrend, da quasi zu einer RUECKKEHR-Adresse zurückgesprungen werden soll, von der man ja gar nicht hergekommen ist:

> Indem die Adresse auf den Stack geschoben wird, wird also ein gedachter Unterprogramm-Aufruf simuliert - was für den Stack ohne Belang ist.

> Sobald nun ein RTS-Befehl ausgeführt wird, liefert der Stack diese Adresse. Und bei dieser Adresse handelt es sich dann um die gewünschte Sprungadresse.

> Dabei ist zu beachten, dass der RTS-Befehl die um 1 niedrigere Adresse als die Rücksprungadresse erwartet:

>> Soll auf die Adresse $3456 gesprungen werden, so müssen also die Werte $34 und $55 auf den Stack geschoben werden.

Diese interessante Methode soll nun veranschaulicht werden:

Angenommen, das X-Register enthält einen Wert zwischen 0 und 5. Abhängig von diesem Wert soll auf eine von sechs verschiedenen Adressen gesprungen werden.

Die gewünschten Adressen sind in einer Tabelle zusammengefasst:
Die niederwertigen Bytes der Adressen sind in einer Tabelle ab $2320 abgelegt und die höherwertigen Bytes in einer zweiten Tabelle ab $2326.

Dabei muss von jeder Adresse der Wert 1 subtrahiert sein, so dass die Tabelle folgendermassen aussieht:

```
2320  41  72  A3  C4  E5  F6  :Low-Bytes
2326  24  25  27  29  2B  2C  :High-Bytes
```

Wenn nun das XR den Wert 0 enthält, so soll nach $2442 gesprungen werden, beim Wert 1 nach $2573, usw.
Dies wird dann im Programm folgendermassen realisiert:

```
LDA $2326,X ;höherwertiger Teil zuerst
PHA
LDA $2320,x ;dann niederwertiger Teil
PHA
RTS         ;Sprung zur Routine
```

Diese Methode ist einfach, schnell und übersichtlich und zudem eine der nützlichsten im Repertoire des 6502-Programmierers.

Im MICROSOFT-BASIC wird sie dazu benutzt, die verschiedenen Routinen wie PRINT, LET, FOR, usw. auszuführen.

Der TIM (Monitor) verwendet sie, um seine Kommandos wie .M, .R, usw. zu interpretieren.

Der STACK-POINTER

Es existiert im 6502-Befehlssatz ein Befehl, der Information über die Stelle gibt, an der sich der Zeiger in den Stapel (=Stackpointer) befindet.
Ebenso gibt es einen Befehl zur Manipulation dieses Zeigers.

Von diesen Möglichkeiten wird man zwar recht selten Gebrauch machen, aber detailliertere Kenntnisse darüber sind dennoch manchmal wertvoll.

> Der Stack ist von oben nach unten organisiert.
> Sobald Informationen auf ihn abgelegt werden, wandert der Stackpointer (SP) nach unten, werden sie zurückgeholt, so steigt er nach oben.
> Gelangt er dabei auf seine niedrigste Adresse $0100, so springt er auf $01FF, sobald eine zusätzliche Information abgelegt werden soll.
>
> Das Programm selbst ist jedoch schon lange vor diesem Punkt gefährdet, da der untere Bereich des Stacks vom Rechner z.B. für die Umwandlung von Zahlen in Strings benötigt wird.

Der Stackpointer zeigt immmer auf die nächste Stelle, die benutzt werden soll.
Hat er den Wert $92, so bedeutet dies, dass er den nächsten Wert auf Adresse $0192 schieben oder von Adresse $0193 holen würde.

Befehle zur Verwendung des Stackpointers

a. Der TXS-Befehl (Transfer XR to Stackpointer = $9A) wird meist dazu benutzt, den gesamten Stack zurückzusetzen. Dabei werden sämtliche in ihm enthaltenen Informationen gelöscht wie beispielsweise die Rücksprungadressen für Unterprogramme und Interrupts.

Beim CBM wird der Pointer auf seine höchste Stelle (=$FA) gesetzt mit:

```
LDX #$FA
TXS
```

b. Der TSX-Befehl (Transfer Stackpointer to XR =$BA) findet vielseitigere Verwendung:

Zum Nachprüfen, ob zuviele Informationen im Stack sind, kann so vorgegangen werden:

```
TSX
CPX #$40
```

Dabei kann selbstverständlich jede andere Abfrage anstatt $40 verwendet werden. Man beachte aber, dass der Wert $40 die niedrigste Stelle beinhaltet, die im CBM verwendet werden darf.

Da der Stack ein Teil des Speichers ist, kann man sich den Inhalt des Stacks auch direkt ansehen, indem man sich den Inhalt der Adressen $0100 bis $01FF anschaut:

Am einfachsten geschieht das mit

```
TSX
LDA $0100,X
```

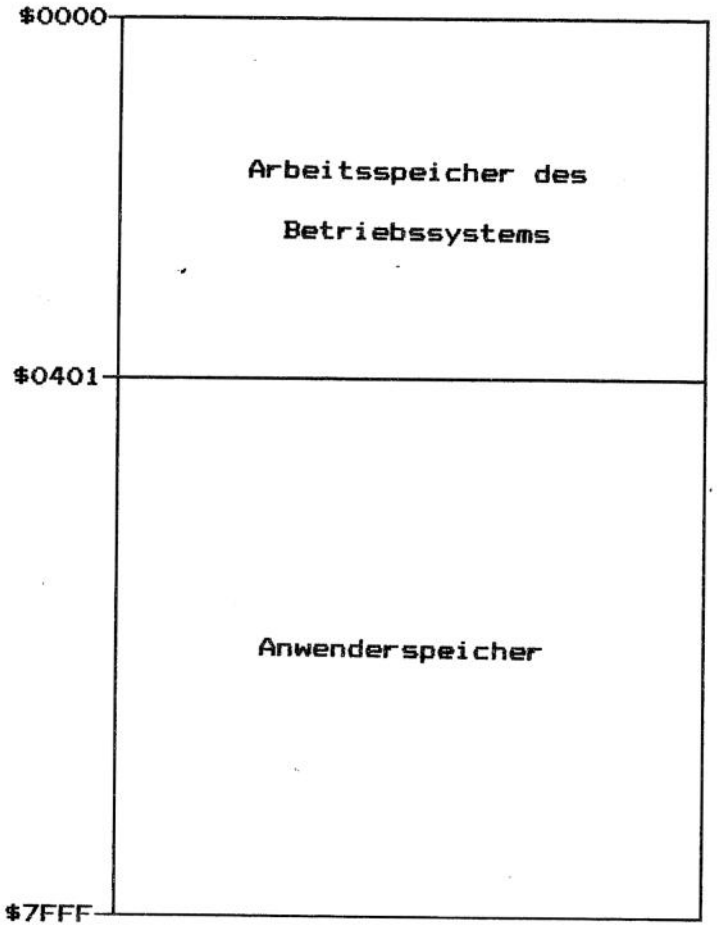
Im AC befindet sich nun der Wert der Stelle im Stack, auf die der Stackpointer (mit dem Wert (XR)) deutet. Durch Erhöhen bzw. Erniedrigen von XR kann der Inhalt beliebiger Adressen im Stack betrachtet werden.

3. Speicherverwaltung

Die Speicherverwaltung ist eine der wichtigsten Aufgaben eines Betriebssystems.
Das Betriebssystem muss hierbei die Aufgabe lösen, den vorhandenen RAM optimal zu nutzen, d.h. der Benutzer-RAM muss unter den verschiedenen Datenarten so aufgeteilt werden, dass kein Speicherplatz ungenutzt bleibt bzw. verschenkt wird.
Dies wird durch eine dynamische Speicherverwaltung ermöglicht, die auch das CBM-Betriebssystem benutzt.

3.1 Speicheraufteilung

Der vorhandene RAM-Bereich ($0000-$7FFF) der COMMODORE-Rechner ist in zwei Teile getrennt:



Dabei ist der Speicherbereich von $0000-$3FFF für die aus der nachstehenden Tabelle ersichtlichen Datenbereiche fest reserviert und nicht veränderbar.

```
$0000-$00FF : Zeropage zur Zwischenspeicherung von
              veränderlichen Betriebssystemdaten
$0100-$1FFF : Stack des Mikroprozessors
$0200-$0250 : BASIC - Input - Puffer
$0251-$025A : Tabelle der Filenummern
$025B-$0264 : Tabelle der Gerätenummern
$0265-$026E : Tabelle der Sekundäradressen
$026F-$0278 : Tastatur - Input - Puffer
$027A-$0339 : Puffer für Rekorder 1
$033A-$03F9 : Puffer für Rekorder 2
```

Der Anwenderbereich jedoch wird dynamisch verwaltet und vom Betriebssystem auf die folgenden vier Datenbereiche aufgeteilt:

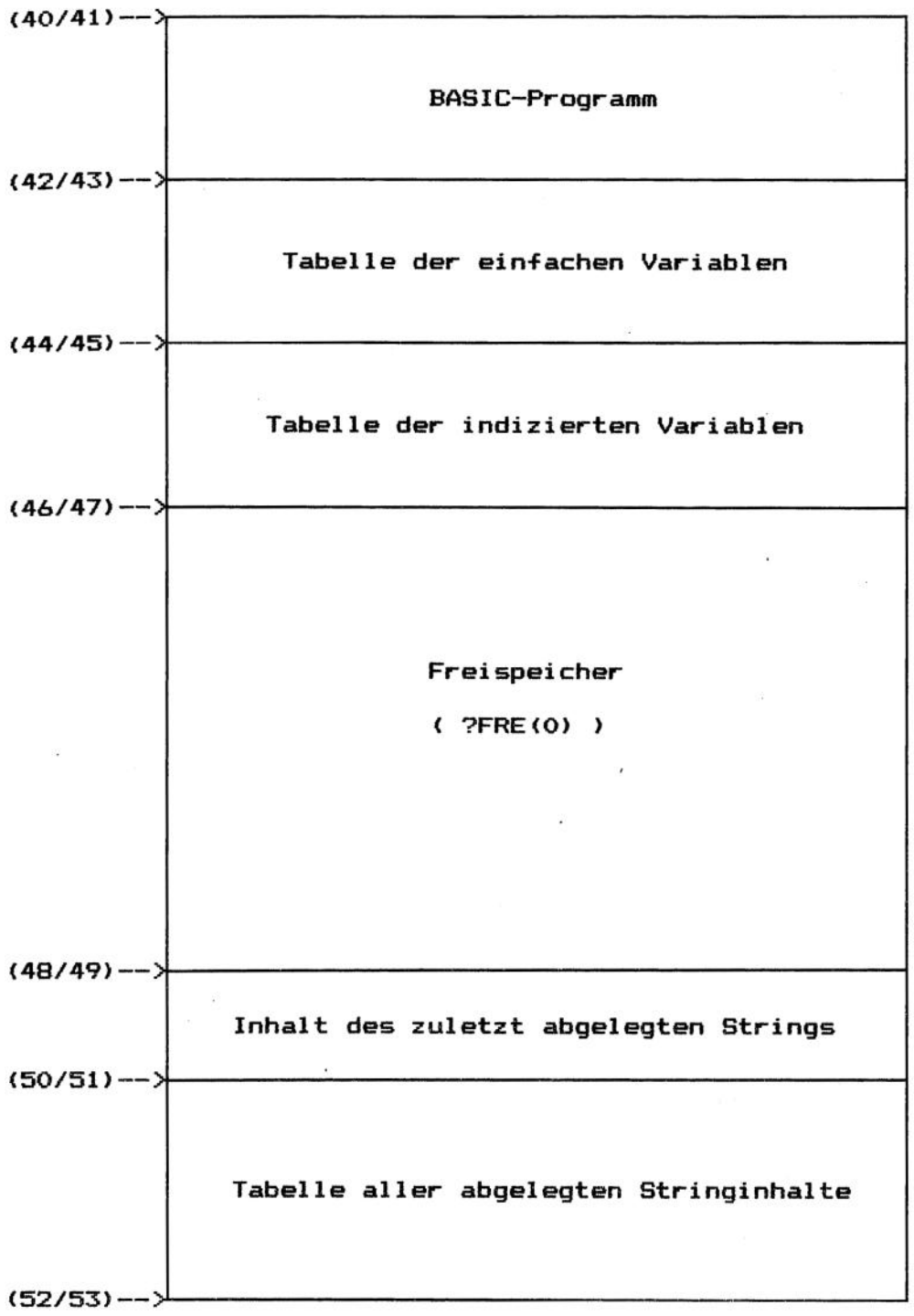
1. BASIC-Zeilen, d.h. Ablage eines BASIC-Programmes.
2. Tabelle der einfachen Variablen.
3. Tabelle der indizierten Variablen (Felder).
4. Tabelle der Stringinhalte.

Um nun eine dynamische Koordination durchführen zu können und eine Ueberlappung dieser vier Bereiche zu vermeiden, benutzt das Betriebssystem zur gegenseitigen Abgrenzung insgesamt sieben Zeiger, die sich in der Zeropage befinden.

Zeigeradresse LB,HB	Bedeutung
40,41	Beginn des BASIC-Programmes
42,43	Anfang der einfachen Variablen
44,45	Beginn der indizierten Variablen
46,47	Ende der Tabelle der einfachen und indizierten Variablen
48,49	Anfang des zuletzt abgelegten Stringinhalts
50,51	Ende des zuletzt abgelegten Stringinhalts
52,53	Obergrenze des Arbeitsspeichers

Die bildliche Darstellung der Aufteilung des Anwenderbereiches auf der nächsten Seite veranschaulicht die Bedeutung dieser Zeiger:

Aufteilung des Anwenderspeicherbereiches:



Die Aufgabe der Zeiger ist es, dem Betriebssystem laufend ein aktuelles Abbild über die Verteilung des Anwenderspeichers auf die vier Datenbereiche zu liefern. Mit Hilfe dieser Zeiger ist es überhaupt erst möglich, ein dynamisches Speicherverwaltungskonzept zu erreichen, da die Vergrösserung eines der vier genannten Bereiche aufgrund von zusätzlichen Daten damit leicht möglich ist, was an einem Beispiel erläutert werden soll:

Hierfür mögen folgende Voraussetzungen gelten:

a. Es sei ein BASIC-Programm gestartet worden, in dem einfache und indizierte Variablen definiert sind.
b. Es werde ein Zeitpunkt im Ablauf des Programmes betrachtet, zu dem bereits eine Variablentabelle (indizierte und nicht indizierte Variablen) angelegt ist.

Wenn nun im weiteren Ablauf des Programmes eine neue einfache Variable angesprochen wird, die in der Variablentabelle noch nicht enthalten ist, so kopiert das Betriebssystem die Arraytabelle, die bei (44/45) beginnt, um soviele Bytes in den Speicher hinauf, wie die hinzugekommene Variable benötigt. Danach werden die Zeiger auf den Beginn (44,45) und das Ende (46,47) der Tabelle der indizierten Variablen wieder richtiggestellt.

Wie man aus diesem Beispiel erkennt, wird fast kein Speicherplatz (siehe 3.2.3) verschenkt. Der vorhandene Speicherplatz ist dann erschöpft, wenn das Ende der Tabelle der Feldvariablen mit der Liste der Stringinhalte, die von der Obergrenze des RAM nach unten hin abgelegt werden, zusammenfällt, d.h. wenn (46/47)=(48/49).

3.2 Speicherablage der verschiedenen Datentypen

3.2.1 Ablage eines BASIC-Programmes

Jede BASIC-Zeile wird folgendermassen im Speicher abgelegt:

a. Die ersten beiden Bytes enthalten die Verkettungsadresse zur nächsten BASIC-Zeile, d.h. die Adresse, an der die nächste BASIC-Zeile beginnt.
b. Die nächsten zwei Bytes (Byte 3 und 4) enthalten die Zeilennummer der BASIC-Zeile.
c. Danach folgt der eigentliche Inhalt einer BASIC-Zeile im ASCII-Code, wobei jedoch die BASIC-Befehle als sogenannte 'Tokens' verschlüsselt werden (siehe Anhang 24) und daher nur ein Byte benötigen (halbcompilierender Interpreter).
d. Jede BASIC-Zeile wird mit einer Null abgeschlossen. Damit ist eine Abtrennung der einzelnen BASIC-Zeilen untereinander erreicht.

Ein BASIC-Programm wird ab der Stelle im RAM abgespeichert, auf die der Zeiger für den Anfang des BASIC-Programmes (40,41) weist. Dies ist üblicherweise die Adresse 1025.
Das komplette BASIC-Programm wird mit drei Nullen abgeschlossen. Die erste Null stellt die Trennung zwischen zwei BASIC-Zeilen und die nächsten beiden Nullen die Verknüpfungsadresse zur nächsten Zeile dar. Da diese aber bei $0000 beginnen würde, erkennt der BASIC-Interpreter damit das Ende des Programmes.
Ein Speicherauszug soll noch einmal die Speicherablage eines kurzen BASIC-Programmes bildlich verdeutlichen:

Programm

```
10 Print A
260 Print B
```

Anmerkung

Alle Angaben über Adressen und Inhalte sind hexadezimal.

Adresse	0401	0402	0403	0404	0405	0406	0407
Inhalt	08	04	0A	00	99	41	00
Bedeut.	Verkettungs-adresse = $0408		Zeilennummer =0*256+10=10		Print	A	Trennungs-zeichen

⇩

Adresse	0408	0409	040A	040B	040C	040D	040E
Inhalt	0F	04	04	01	99	42	00
Bedeut.	Verkettungs-adresse = $040F		Zeilennummer =1*256+4= =260		Print	B	Trennungs-zeichen

⇩

Adresse	040F	0410	0411	0412	0413	0414	0414
Inhalt	00	00	?	?	?	?	?
Bedeut.	Verkettungs-adresse = $0000		Beginn der Variablentabelle				

3.2.2 Kennzeichnung von Variablennamen

Bevor auf die Speicherablage von einfachen und indizierten Variablen eingegangen wird, soll ihre Namenskennzeichnung betrachtet werden.
Für den Variablennamen sind zwei Bytes reserviert. Die Verschlüsselung geschieht nach folgenden Regeln:

a. Die beiden Namenszeichen werden durch ihren ASCII-Code dargestellt. Wenn der Name jedoch nur aus einem Buchstaben besteht, wird anstelle des zweiten Zeichens der Wert 0 genommen.
b. Der Variablentyp wird mit in die Namensverschlüsselung eingebunden. Hierbei gilt nachstehende Zuordnung.

Variablentyp	1.Zeichen	2.Zeichen
Gleitkommavariable	ASCII-Code	ASCII-Code
Integervariable	ASCII-Code+$80	ASCII-Code
Stringvariable	ASCII-Code	ASCII-Code+$80

Beispiele

Variablenname	1. Byte	2 .Byte
A	$41	$00
A%	$41+$80=$C1	$00+$80=$80
A$	$41	$00+$80=$80
C8	$43	$38
C8%	$43+$80=$C3	$38+$80=$B8
C8$	$43	$38+$80=$B8

3.2.3 Ablage von einfachen Variablen

Die Tabelle der einfachen Variablen beginnt unmittelbar nach einem BASIC-Programm, d.h. im Anschluss an die drei Nullen am Ende der letzten BASIC-Zeile. Der Zeiger auf den Anfang der einfachen Variablen in 42,43 weist auf diese Stelle.
Die Variablen werden in 7-Byte Feldern in der Reihenfolge ihres ersten Auftritts im BASIC-Programm abgelegt. Die ersten beiden Bytes enthalten den Variablennamen (siehe 3.2.2) und die nächsten fünf Bytes den Inhalt der Variablen.
Drei Variablentypen müssen dabei unterschieden werden:

1. Gleitkommavariablen
2. Integervariablen
3. Stringvariablen

Die fünf Bytes, die für den Inhalt reserviert sind, werden nur teilweise verbraucht, wie an folgender Aufstellung zu erkennen ist:

a. Gleitkommavariablen:
   Die Information benötigt 5 Bytes
   (siehe Zahlendarstellung)
   Gesamt: Name + Information = 2 + 5 = 7 Bytes

b. Integervariablen:
   Die Information benötigt 2 Bytes
   Gesamt: Name + Information = 2 + 2 = 4 Bytes
   => 3 Bytes werden verschenkt!

c. Stringvariablen:
   Die Information benötigt 3 Bytes:
   Das erste der drei Bytes enthält die Länge des jeweiligen Strings (0-255), die nächsten beiden Bytes den Zeiger auf den Beginn des Strings.Der eigentliche Stringinhalt ist im oberen Bereich des RAM zu finden (siehe 3.2.5).
   Gesamt: Name + Information = 2 + 3 = 5 Bytes
   => 2 Bytes werden verschenkt!

Die freien Bytes werden in den beiden letztgenannten Fällen mit Nullen aufgefüllt.
Wie man sieht, wird trotz des dynamischen Speicherprinzips der verfügbare Speicherplatz nicht vollständig ausgenutzt.
Man darf jedoch dabei auch nicht übersehen, dass dadurch der Zugriff auf eine einfache Variable verkürzt wird, da der BASIC-Interpreter die Tabelle mit einer konstanten Sprungweite nach einer bestimmten Variablen durchsuchen kann.

```
Beispiel    10 C8=1
--------    20 C8%=255
            30 C8$="ABC
            40 END
```

Nach RUN sieht die Tabelle der einfachen Variablen wie folgt aus:

Inhalt	43	38	81	00	00	00	00
Bedeut.	"C" "8" = C8		= + 1				

Inhalt	C3	B8	00	FF	00	00	00
Bedeut.	"C"+80 "8"+80 = C8%		= 255		nicht benutzt		

Inhalt	43	B8	03	1F	04	00	00
Bedeut.	"C" "8"+80 = C8$		String-länge =3	Zeiger auf Stringinhalt => $041F		nicht be-nutzt	

3.2.4 Ablage von indizierten Variablen

Die einfachen Variablen sind im 7-Byte-Raster abgelegt und verschenken dadurch im Falle der Integer- bzw. Stringvariablen Platz im Speicher.
Im Gegensatz dazu wird bei der Ablage von indizierten Variablen (Feldern) der Speicherplatz vollständig genutzt.
Dies erfordert jedoch auch einen etwas komplizierteren Zugriff.
Die Ablage der Felder erfolgt wie bei den einfachen Variablen in der Reihenfolge ihres Auftretens (ohne vorherige Dimensionierung) bzw. in der Reihenfolge der DIM-Anweisungen.
Hierbei sollte beachtet werden, dass beim ersten Auftreten einer indizierten Variablen ohne vorherige Dimensionierung automatisch ein Feld mit elf Elementen reserviert wird, d.h. A$(1) ohne DIM A$(X) bewirkt die gleiche Vereinbarung wie DIM A$(10). Wenn also weniger als elf indizierte Variable des gleichen Namens in einem Programm benötigt werden, so kann durch eine DIM-Anweisung, die in diesem Fall ja nicht zwingend ist, Speicherplatz eingespart werden.

Grober Aufbau eines Feldes

Gliederung des Feldkopfes

Byte 1 und 2 :	Name des Feldes (Namenskennzeichnung siehe 3.2.2)
Byte 3 und 4 :	Länge des benötigten Speicherbereiches für das ganze Feld (einschliesslich Feldkopf).
Byte 5 :	Anzahl der Dimensionen des Feldes.
Byte 6,7 - ? :	Anzahl der Elemente pro Dimension.Für jede Dimension werden zwei Bytes benötigt, da ja mehr als 255 Elemente erlaubt sind. Die zuletztgenannte Dimension in der DIM-Anweisung wird zuerst aufgeführt.

Beispiel 10 DIM A$(1,2)

Zugehöriger Feldkopf

Inhalt	41	80	1B	00	02	00	03	00	02
Bedeut.	"A" 00+80 Feldname A$		Benötigte Speicher-menge L,H		Anz. der Dim. = 2	Anz. der Elemente der 2.Dim. = 3		Anz. der Elemente der 1.Dim. = 2	

Anordnung der Feldelemente

An den Feldkopf schliessen sich direkt die Feldelemente an. Bei eindimensionalen Feldern werden die einzelnen Elemente in der Reihenfolge ihrer Feldnummern abgelegt.

Beispiel DIM A$(3)

Feldkopf	A$(0)	A$(1)	A$(2)	A$(3)

Bei zweidimensionalen Feldern werden die Feldelemente spaltenweise abgelegt.

Beispiel DIM A$(1,2)

Feldkopf	A$(0,0)	A$(1,0)	A$(0,1)	A$(1,1)	A$(0,2)	A$(1,2)

Der Inhalt einer indizierten Variablen wird genauso wie die einfachen Variablen im Speicher dargestellt.
Der Unterschied besteht jedoch darin, dass eine Integervariable nur vier bzw. eine Stringvariable nur fünf Bytes benötigt.
Die Grösse des Speicherplatzes, den ein Feld einnimmt, errechnet sich wie folgt:

```
Bytemenge = Anzahl Bytes pro Element * Anzahl der Elemente
                                + 2 * Anzahl der Dimensionen
                                + 5 Bytes
```

Ein abschliessendes Beispiel soll die Abspeicherung von Feldern veranschaulichen:

BASIC-Programm

```
10 DIM A(1)
20 DIM A%(1)
30 DIM A$(1)
40 A(1)=3
50 A%(0)=1:A%(1)=256
60 A$(0)="ABC":A$(1)="DEF"
```

Nach RUN zeigt der Zeiger in (44,45) auf den Beginn der Felder.
Die Speicherinhalte ab dieser Adresse sehen dann folgendermassen aus:

Feldkopf

Inhalt	41	00	11	00	01	00	02
Bedeut.	Feldname = A		Benötigt. Speicher-platz = 17		Anz. der Dim. = 1	Anz. der Elemente der Dim. = 2	

Tabelle der Feldelemente

Inhalt	00	00	00	00	00	82	40	00	00	00	
Bedeut.	Feldelement A(0) nicht belegt					Feldelement A(1) = + 3					⇨

Feldkopf

Inhalt	C1	80	0B	00	01	00	02
Bedeut.	Feldname = A%		Benötigt. Speicher-platz = 11		Anz. der Dim. = 1	Anz. der elemente der Dim. = 2	

Tabelle der Feldelemente

Inhalt	00	01	01	00	
Bedeut.	Feldelement A%(0) = +1		Feldelement A%(1) = +256		⇨

Feldkopf

Inhalt	41	80	0d	00	01	00	02
Bedeut.	Feldname = A$		Benötigt. Speicher-platz = 13		Anz. der Dim. = 1	Anz. der elemente der Dim. = 2	

Tabelle der Feldelemente

Inhalt	03	50	04	03	5C	04
Bedeut.	Feldelement A$(0)			Feldelement A$(1)		
	String-länge =3	Zeiger auf Stringinhalt => $0450		String-länge =3	Zeiger auf Stringinhalt => $045C	

3.2.5 Ablage der Stringinhalte

Sowohl bei einfachen als auch bei indizierten Variablen wird der Stringinhalt nicht direkt abgespeichert. In der Variablentabelle ist nur die Länge des Strings und der Zeiger auf den zugehörigen Stringinhalt enthalten. Dieser zeigt auf die Stelle, an welcher der jeweilige Stringinhalt beginnt.
Die Stringinhalte stehen irgendwo zwischen dem Ende der Arraytabelle (46,47) und der obersten RAM-Grenze (52,53).
Es soll nun erklärt werden, wie die Strings in diesem Bereich abgelegt werden:

1. Die Inhalte der Strings werden normalerweise (siehe 3.2.6), bei der Obergrenze des RAM (52,53) beginnend, nach unten hin abgespeichert.

2. Der allererste Stringinhalt in einem BASIC-Programm wird so abgelegt, dass er genau an der Obergrenze des verfügbaren Arbeitsspeichers endet. Sein Anfang wird dann in 48,49 und sein Ende+1 in 50,51 zwischengespeichert. Beim ersten String wird also der Zeiger auf das Ende des letzten Strings mit dem TOP-OF-RAM-Zeiger in 52,53 gleichgesetzt. Der nächste Stringinhalt wird so abgespeichert, dass er genau vor dem Beginn des zuletzt abgelegten String ((48/49)-1) endet. Sein Anfang wird dann wieder in 48,49, sein Ende+1 in 50,51 gespeichert, d.h. der Zeiger in 50,51 wird jetzt mit dem vorherigen Inhalt des Zeigers in 48,49 gleichgesetzt. Dies wird fortgesetzt, solange die Liste der Stringinhalte noch nicht in die Arraytabelle hineinwächst, d.h. solange (48/49) grösser ist als (46/47).

Interessant ist, dass bei einer Stringzuweisung, Stringverkettung oder Stringoperation immer ein neuer Platz für den Stringinhalt bereitgestellt wird, auch wenn der neue Stringinhalt kürzer ist als der alte oder wenn gar mehrere völlig identische Zuweisungen im Programm vorkommen.

Beispiel

```
10  A$="ABC"
20  B$=A$+"D"
30  B$=A$+"D"
40  B$=A$+"D"
```

Bei diesem Beispiel werden für den String B$ zwölf Bytes verbraucht, obwohl die drei letzten BASIC-Zeilen dem B$ jeweils die gleiche Zeichenkette zuweisen. Das Betriebssystem prüft also nicht die Uebereinstimmung von Stringzuweisungen an dieselbe Stringvariable, sondern teilt jedem Stringinhalt, der während des Programmlaufes angesprochen wird, immer wieder einen neuen Platz zu.
Dies führt dazu, dass der Speicherplatz für die Stringinhalte unter Umständen relativ rasch aufgebläht wird. Ehe sich aber die Arraytabelle und die Tabelle der Stringinhalte überlappen, sucht eine spezielle Systemroutine, die sogenannte GARBAGE-COLLECT-Routine ("Müllsuchroutine" GARBA2= 54272 < 50794), alle Zeiger der Stringvariablen in der gesamten Variablentabelle durch, um festzustellen, welche Stringinhalte keine Gültigkeit mehr besitzen. Die gültigen Stringinhalte, d.h. die Stringinhalte, auf die ein Zeiger verweist, werden dann zusammengeschoben und in den oberen RAM-Bereich hinaufkopiert, wodurch wieder freier Speicherplatz entsteht. Die GARBAGE-COLLECT-Routine kann durch den BASIC-Befehl PRINT FRE(0) erzwungen werden. Sie kann im Extremfall über 20 Minuten benötigen. Der Rechner ist in dieser Zeit scheinbar tot, da er auch nicht mit der STOP-Taste unterbrochen werden kann.
Das folgende Programm zeigt eine solche Verzögerung auf, die durch die GARBAGE-COLLECT-Routine erzeugt wird:

```
10  DIM C$(350)
20  A$="ABCD":B$="DEFG"
30  FOR I=1 TO 350
40  C$(I)=A$+B$
50  NEXT
60  FOR I=1 TO 350
70  C$(I)=A$+B$
80  NEXT
90  PRINT"START"
100 PRINT FRE(0)
110 PRINT "ENDE"
120 END
```

Bei BASIC 3 benötigt die GARBAGE-COLLECT-Routine in diesem Programm ungefähr zehn Sekunden, um im Stringbereich "aufzuräumen". Bei BASIC 4 wurde eine Beschleunigung der GARBAGE- COLLECT- Routine dadurch erreicht, daß hinter dem eigentlichen Inhalt des Strings ein Pointer angelegt wird, der auf die Adresse der Länge des Strings innerhalb des Stringdescriptors zeigt. Die Stringinhalte werden bei einer Neuzuweisung zu einem schon angelegten String- wie auch im BASIC 3- vom oberen Ende des Arbeitsspeichers her angelegt. Der Stringdescritor selbst wird jedoch nicht neu angelegt, sondern nur auf die neuen Parameter geändert. Die GARBAGE-COLLECT- Routine konnte dadurch so schnell gemacht werden, dass sie beim CBM 8000 kaum noch bemerkbar ist.

3.2.6 Besonderheiten bei Stringvariablen

Wie bereits ausführlich erläutert, wird der Inhalt von Stringvariablen normalerweise im oberen RAM-Bereich abgespeichert.
Es gibt jedoch einige spezielle Ausnahmefälle, bei denen von diesem Verfahren abgewichen wird:

a. Eine Stringvariable erhält eine Stringkonstante zugewiesen.

   z.B. 10 A$="ABC"

   => Der Zeiger der Stringvariablen zeigt auf die Stelle, an der die Konstante im Programmtext steht.

b. Einer Stringvariablen wird ein DATA-Element zugewiesen.

   z.B. 10 READ C$
        20 DATA "TEST"

   => Der Zeiger der Stringvariablen zeigt auf das entsprechende Datenelement in der BASIC-Zeile.

c. Wie a. oder b. mit zusätzlicher Zuweisung an eine weitere Stringvariable.

   z.B. 10 A$="ABC"
        20 B$=A$

   oder

        10 READ C$
        20 B$=C$
        30 DATA "TEST"

   Auch hier weist der Stringzeiger auf die Stringkonstante bzw. auf das DATA-Element im BASIC-Text.

A. Prüfung, ob Platz im Speicher	REASON	49960 < 45984

Voraussetzungen

Im AC,YR muss die Adresse LB,HB stehen, bis zu der Speicherplatz benötigt wird.

Ablauf der Routine

Ist die angegebene Adresse kleiner als die Anfangsadresse der Stringinhalte (48,49), erfolgt Rücksprung.
Ansonsten wird die GARBAGE-COLLECT-Routine aufgerufen:
Nach nochmaliger überprüfung, ob der Platz bis zu der gewünschten Adresse jetzt frei ist, erfolgt entweder die Fehlermeldung " OUT OF MEMORY ERROR " oder Rücksprung.

Diese Routine kann man benutzen, um zu überprüfen, ob ein bestimmter Speicherbereich, den man z.B. für die Ablage von speziellen Daten benötigt, mit den Stringinhalten kollidiert.

B. NEW	SCRTCH	50525 < 46548

Voraussetzungen

Keine

Ablauf der Routine

Die Wirkung ist die des BASIC-Befehls NEW:

a. In die ersten beiden Stellen des Programmspeichers (Beginn in 40,41) wird eine Null geschrieben.
b. Der Variablenanfangszeiger (42,43) wird auf den Wert des Programmanfangzeigers (40,41) + 2 gesetzt.
c. Der Programmzeiger in 119,120 (für die CHRGET-Routine) wird auf den Wert des Programmanfangzeigers - 1 gesetzt.
d. Der Stringanfangszeiger (48,49) wird mit dem RAM-Obergrenze-Zeiger (52,53) gleichgesetzt.
e. Alle Files und Kanäle werden geschlossen.
f. Der Feldvariablenanfangszeiger (44,45) und der Variablenendezeiger (46,47) werden auf den Wert des Variablenanfangzeigers (42,43) gesetzt.
g. Der DATA-Zeiger wird auf den Anfang des Programmspeichers gerichtet.

C. CLR	CLEARC	50553 < 46576

Siehe NEW ab Punkt d.

D. RESTORE	RESTOR	50992 < 47031

Voraussetzungen

Keine

Ablauf der Routine

Der DATA-Zeiger in 62,63 wird zurückgesetzt, d.h. er erhält den Wert des Programmanfangzeigers (40/41) - 1.

E. Umwandlung eines Zeilennummer-String in den 2-Byte-Kode	LINGET	51315 < 47350

Eine Zeilennummer im ASCII-Format wird in den 2-Byte-Kode umgewandelt. Dieser Kode entspricht der Ablage einer BASIC-Zeilennummer im Programmspeicher.

Wichtige Speicherzellen

Adresse	Bedeutung
17,18=$11,12	Zeilennummer im 2-Byte-Kode
119,120=$77,78	CHRGET-Pointer (LB,HB)

Voraussetzungen
---
1. Das erste Zeichen des Strings muss im AC stehen.
2. Der Programmzeiger (CHRGET-Pointer in 119,120) muss auf die Anfangsadresse des Strings (= Adresse des ersten Zeichens) gerichtet werden.
3. Der Zeilennummer-String muss mit einem nicht-numerischen ASCII-Zeichen enden.

Ablauf der Routine

Der Wert des Strings bis zum ersten nichtnumerischen Zeichen steht nach dem Rücksprung als 2-Byte-Kode in 17,18 (LB,HB).
Wird die höchstzulässige Zeilennummer überschritten, erfolgt "SYNTAX ERROR".
Wird versucht, einen String umzuwandeln, dessen 1.Zeichen keine Zahl ist, so wird in 17,18 eine Null eingeschrieben.

F. Berechnung der Startadresse einer Programmzeile — FNDLIN — 50476 < 46499

Wichtige Speicherzellen

Adresse	Bedeutung
17,18=$11,12	Zeilennummer im 2-Byte-Kode
92,93=$5C,5D	Startadresse der BASIC-Zeile

Voraussetzungen

Die Zeilennummer der zu suchenden BASIC-Zeile muss als 2-Byte-Kode in 17,18 (LB,HB) stehen.

Ablauf der Routine

Die Routine durchsucht den BASIC-Textspeicher von Anfang an nach dieser Zeile.
Dabei wird ein Vergleich der vorgegebenen Zeilennummer in 17,18 mit der jeweiligen Zeilennummer des BASIC-Programms durchgeführt. Die Suche wird abgebrochen, wenn die gesuchte Zeilennummer kleiner ist als die der gerade untersuchten BASIC-Zeile oder wenn das Programmende erreicht ist (High-Byte der Verkettungsadresse =0).
War die Zeile vorhanden, so steht nach dem Rücksprung ihre Startadresse in 92,93 und das CARRY-Bit ist gesetzt.
Bei einer erfolglosen Suche steht in 92,93 (LB,HB) die Startadresse der nächsten Programmzeile bzw. die Programmendadresse und das CARRY-Bit ist gelöscht.

Bemerkungen zum Beispiel von E und F

Zugehöriges BASIC-Programm:

```
10 A=0
20 TESTZEILE
30 REM
40 END
```

Aufgabenstellung

Es soll eine Zeilennummer (hier Zeile 20), die als String im ASCII-Format ab $027A ff. steht, in den 2-Byte-Kode umgewandelt werden und anschliessend die Startadresse dieser Zeile berechnet werden.

Lösung

Nach Einschreiben der Zeilennummer in $027A-$027B und Aufruf der beiden Routinen steht ihre Anfangsadresse in $5C,5D.
Zur Kontrolle wird der Inhalt der Zeile 20 auf dem Bildschirm ausgegeben.

```
                    0001              ;BEISPIEL ZU E. UND F.:
                    0002              ;----------------------
                    0003              .OS
                    0004              .BA $027A      ;ABLAGE DES
027A- 32 30         0005              .BY '20'       ;NUMERISCHEN STRINGS
027C- FF            0006              .BY $FF        ;ENDE-ZEICHEN
                    0007              ;----------------------
                    0008              .BA $033A
                    0009 STRINGANF    .DE $027A
                    0010 STRINGEND    .DE STRINGANF+2
                    0011 ZEIG         .DE $5C        ;ZEIGT AUF ADRESSE DER
                    0012              ;                BASIC-ZEILE
                    0013 CHRGET       .DE $77        ;CHRGET-POINTER
                    0014              ;----------------------
033A- A9 7A         0015 ANFANG       LDA #L,STRINGANF
033C- 85 77         0016              STA *CHRGET
033E- A9 02         0017              LDA #H,STRINGANF
0340- 85 78         0018              STA *CHRGET+1
0342- AD 7A 02      0019              LDA STRINGANF        ;1.ZEICHEN DER
                    0020              ;                     ZEILENNUMMER IN AC
0345- 20 73 C8      0021              JSR $C873   (<B8F6) ;WANDELT ZEILEN-
                    0022              ;             NUMMER UM IN 2-BYTE-CODE
0348- 20 2C C5      0023              JSR $C52C   (<B5A3) ;SUCHT DIE START-
                    0024              ;             ADRESSE DER ZEILE
034B- 20 3F E5      0025              JSR $E52F   (<E3B6) ;ZEILENVORSCHUB
034E- 18            0026              CLC
034F- A5 5C         0027              LDA *ZEIG            ;AUSGABE DES
0351- 69 04         0028              ADC #$04             ;ZEILEN-
0353- A4 5D         0029              LDY *ZEIG+1          ;INHALTES
0355- 90 01         0030              BCC LABEL            ;AUF DEN
0357- C8            0031              INY                  ;BILDSCHIRM
0358- 20 1C CA      0032 LABEL        JSR $CA1C   (<BB1D) ;AUSDRUCKROUTINE
035B- 20 2F E5      0033              JSR $E52F   (<E3B6) ;ZEILENVORSCHUB
035E- 00            0033              BRK
                    0034              .EN

LABEL FILE:   ( / = EXTERNAL )

/STRINGANF=027A          /STRINGEND=027C          /ZEIG=005C
/CHRGET=0077             ANFANG=033A              LABEL=0358
```

G. Suche nach einer einfachen Variablen	53193 < 49543

Folgende Speicherzellen der Zeropage dienen als Schnittstelle zu dieser Routine:ln2

Adresse	Bedeutung
66=$42	1. Zeichen des Variablennamens
67=$43	2. Zeichen des Variablennamens
68=$44	Variablenadresse LB
69=$45	Variablenadresse HB

Voraussetzungen

Der Name der einfachen Variablen, die gesucht werden soll (Verschlüsselung siehe 3.2.2), muss in 66,67 abgelegt werden.

Ablauf der Routine

Die Routine durchsucht die gesamte Tabelle der einfachen Variablen mit einer Schrittweite von sieben Bytes.
Wird die Variable gefunden, so übergibt die Routine deren Adresse in 68,69. Dabei zeigt der Zeiger in 68,69 auf das erste Datenbyte der Variablen, d.h. also auf das dritte Byte des 7-Byte-Feldes.
Wenn die Variable noch nicht existiert, wird sie am Ende der Tabelle der einfachen Variablen angehängt und die Tabelle der Felder wird um sieben Bytes nach oben geschoben. Die fünf Datenbytes der neuen Variablen werden auf Null gesetzt. Auch in diesem Fall zeigt der Suchzeiger dann wieder auf das erste Byte hinter dem Namen der neuangelegten Variablen.
Weiterhin fängt diese Routine die beiden Systemvariablen TI und ST ab, d.h. bei einer versuchten Wertzuweisung an eine dieser Variablen in BASIC wird "SYNTAX ERROR" gemeldet.

H. Suche nach einer Feldvariablen 53491 < 49987

Wichtige Speicherzellen

Adresse	Bedeutung
5=$05	Anzahl der Dimensionen
6=$06	DIM-Flag
66=$42	1. Zeichen des Variablennamens
67=$43	2. Zeichen des Variablennamens
68=$44	Variablenadresse LB
69=$45	Variablenadresse HB

Voraussetzungen

Da die Routine eine Unterroutine der gesamten Variablenverwaltung ist, müssen vor ihrem Aufruf einige Bedingungen erfüllt sein, damit sie überhaupt benutzt werden kann und Fehlermeldungen vermieden werden:

1. Das DIM-Flag in 6 muss auf Null gesetzt werden.
2. Die Zahl der Dimensionen in 5 muss auf Eins gesetzt werden, da nur eindimensionale Feldvariablen gesucht werden können.
3. Der Name der Feldvariablen, die gesucht werden soll (Verschlüsselung siehe 3.2.2), muss in 66,67 abgelegt werden.
4. Der Index des zu suchenden Feldelementes muss auf den Stapel geschoben werden: zuerst das höherwertige, dann das niederwertige Byte.
5. Die Routine muss mit JMP aufgerufen werden, da bei einem Aufruf mit JSR ein falscher Index geholt würde. Deshalb muss vor der Ablage des Index auf den Stapel noch die Rücksprungadresse - 1 manuell auf den Stapel geschoben werden.

Ablauf der Routine

Die Tabelle der Felder, auf deren Beginn der Zeiger in 44,45 weist, wird von Anfang an nach dem Feldnamen durchsucht. Den Beginn des folgenden Feldes kann die Routine aufgrund der Angabe der benötigten Bytes im Feldkopf errechnen. Wird nun der entsprechende Feldname bei der Durchsuchung der Tabelle der Felder gefunden, so wird die Anfangsadresse des gewünschten Feldelementes berechnet und in 68,69 übergeben. Bei einer erfolglosen Suche wird automatisch ein Feld (mit dem angegebenen Feldnamen) mit elf Elementen an die Array-Tabelle angehängt. Die Feldbytes werden danach mit Nullen gefüllt. Der Zeiger in 68,69 zeigt dann wieder auf das angegebene Feldelement.

I. Variablensuche mit Variablen- namen aus dem BASIC-Text	PTRGET	53101 < 49451

Will man den Namen einer zu suchenden Variablen (indiziert und nicht indiziert) direkt aus dem BASIC-Text übergeben, so erleichtert diese Routine die Arbeit erheblich. Denn sie holt sich selbstständig aus der entsprechenden BASIC-Zeile den Variablennamen und bei Feldern den Index.
Die CHRGET-Routine übernimmt hierbei das Lesen der einzelnen Bytes des BASIC-Textes aus dem Programmspeicher.

Wichtige Speicherzellen

Adresse	Bedeutung
7=$07	Flag für Variablenart ($00=numerisch,$FF=String)
8=$08	Flag für Integer ($80=Integer,$00=sonst)
66=$42	1. Zeichen des Variablennamens
67=$43	2. Zeichen des Variablennamens
68=$44	Variablenadresse LB
69=$45	Variablenadresse HB
92=$5C	Zeiger auf Feldkopf LB
93=$5D	Zeiger auf Feldkopf HB

Voraussetzungen

1. Hinter dem SYS-Befehl im BASIC-Programm muss, durch Komma getrennt, die gewünschte Variable angegeben werden.
2. Im aufgerufenen Maschinenprogramm muss zuerst das Komma überlesen werden (sonst "SYNTAX ERROR"), was durch den einmaligen Aufruf der CHRGET-Routine mit JSR $0070 erreicht wird.

Ablauf der Routine

Die Routine verschlüsselt zuerst den durch das BASIC-Programm übergebenen Variablennamen (siehe 3.2.2) und springt danach in die entsprechenden Suchroutinen (siehe G oder H). Nach dem Rücksprung befindet sich der verschlüsselte Variablenname in 66,67, die Variablenadresse in 68,69 und die Anfangsadresse des Feldkopfes in 92,93. Bei erfolgloser Suche wird eine neue Variable bzw. ein neues Feld (mit elf Elementen) angelegt. Weiterhin können die zwei Flags in 7 und 8 abgefragt werden, ob die jeweilige Variable vom richtigen Typ ist.

Anmerkung: Bei indizierten Variablen kann der Index ein allgemeiner, numerischer Ausdruck sein.
Der Index kann also seinerseits wiederum eine Variable enthalten.

Drei Assemblerbeispiele sollen die Vorgehensweise bei der Variablensuche und den Zugriff auf den Variableninhalt deutlich machen.

Hierzu einige Bemerkungen:
==========

Beispiel zu G

Aufgabenstellung

Es soll die Anfangsadresse der Zahlenvariablen C8 gesucht werden. Diese Variable sei vorher in nachstehendem BASIC-Programm definiert worden:

```
10 C8=1
20 END
```

Lösung

Nach der Uebergabe des Variablennamens in $42,43 und nach Aufruf der Suchroutine steht in $44,45 die Adresse des ersten Datenbytes.
Zur Kontrolle wird diese Adresse auf dem Bildschirm ausgegeben.

Beispiel zu H

Zugehöriges BASIC-Programm:

```
10 DIM A$(5)
20 A$(1)="TEST1"
30 A$(4)="TEST4"
40 END
```

Aufgabenstellung

Es soll nun z.B. auf die Stringvariable A$(1) zugegriffen werden und ihr Inhalt an eine beliebige Stelle im RAM (hier $027A ff.) umkopiert werden.

Lösung

Nach der Uebergabe des Variablennamens in $42,43 und nach Erfüllen der Einsprungvoraussetzungen befindet sich nach dem Rücksprung in $44,45 der Zeiger auf das 1.Datenbyte (=Stringlänge) von A$(1). Die Stringlänge und die Stringadresse werden dann nach $4D-$4F umgespeichert und mittels indirekt indizierter Adressierung wird der Stringinhalt gelesen und nach $027A ff. kopiert.
Zur Kontrolle wird der gefundene Stringinhalt auf dem Bildschirm geschrieben.
Wird ein String gesucht, der keinen Inhalt besitzt (Stringlänge=0), so erfolgt keine Ausgabe.

Beispiel zu I

Zugehöriges BASIC-Programm:

```
10 SYS 826,B%
20 END
```

Aufgabenstellung

Die Maschinenroutine, die mit SYS aufgerufen wird, soll eine Integerzahl, die z.B. in $027A,$027B (HB,LB) steht, der Variablen B% zuweisen.

Lösung

Nach dem Rücksprung aus der entsprechenden Suchroutine steht wieder in $42,43 die Adresse des ersten Datenbyte, in diesem Fall die Adresse des High-Byte der Integerzahl. Mittels indirekt indizierter Adressierung wird die Integerzahl an diese Stelle transferiert.
Mit PRINT B% kann man sich dann vom Erfolg überzeugen, wenn man vor dem Start des BASIC-Programms eine bestimmte Integerzahl in $027A,027B abgelegt hat.

```
                    0001              BEISPIEL ZU G.
                    0002             ;-------------------------------
                    0003 NAME        .DE $42       ;VARIABLENNAME
                    0004             ;
                    0005 ADRESSE     .DE $44       ;VARIABLENADRESSE
                    0006             ;-------------------------------
                    0007             ;
                    0008             .BA $033A
033A- A9 43         0009 ANFANG      LDA #'C'
033C- 85 42         0010             STA *NAME
033E- A9 38         0011             LDA #'8'
0340- 85 43         0012             STA *NAME+1
0342- 20 C9 CF      0013             JSR $CFC9   (<C187) ;SUCHT VARIABLE C8
0345- 20 2F E5      0014             JSR $E52F   (<E3B6) ;ZEILENVORSCHUB
0348- A5 44         0015             LDA *ADRESSE        ;AUSGABE DER
034A- 85 FB         0016             STA **$FB           ;ADRESSE DES ERSTEN
034C- A5 45         0017             LDA *ADRESSE+1      ;DATENBYTES AUF
034E- 85 FC         0018             STA **$FC           ;BILDSCHIRM AUSGEBEN
0350- 20 6A E7      0019             JSR $E76A           ;=AUSGABEROUTINE
0353- 20 2F E5      0020             JSR $E52F   (<E3B6) ;ZEILENVORSCHUB
0356- 00            0021             BRK
                    0022             .EN

LABEL FILE:   ( / = EXTERNAL )

/NAME=0042              /ADRESSE=0044           ANFANG=033A
```

```
                    0001            ;BEISPIEL ZU H.
                    0002            ;-------------------------------
                    0003 ANZAHL     .DE $05       ;ANZAHL DER DIMENSIONEN
                    0004 DIMFLAG    .DE $06
                    0005 NAME       .DE $42       ;VARIABLENNAME
                    0006 ADRESSE    .DE $44       ;VARIABLENADRESSE
                    0007 PUFFER     .DE $4D
                    0008            ;-------------------------------
                    0009            .BA $033A
033A- A9 00         0010 ANFANG     LDA #$00
033C- 85 06         0011            STA *DIMFLAG
033E- A9 01         0012            LDA #$01              ;ANZAHL DER
0340- 85 05         0013            STA *ANZAHL           ;DIMENSIONEN =1
0342- A9 03         0014            LDA #H,SPRUNG-1       ;RUECKSPRUNG-
0344- 48            0015            PHA  ;                 ADRESSE MANUELL
0345- A9 58         0016            LDA #L,SPRUNG-1       ;AUF STACK
0347- 48            0017            PHA  ;                 SCHIEBEN
0348- A9 00         0018            LDA #$00              ;GEWUENSCHTER
034A- 48            0019            PHA  ;                 FELDINDEX=1
034B- A9 01         0020            LDA #$01              ;AUF STACK
034D- 48            0021            PHA  ;                 SCHIEBEN.
034E- A9 41         0022            LDA #'A'
0350- 85 42         0023            STA *NAME
0352- A9 80         0024            LDA #$80
0354- 85 43         0025            STA *NAME+1
0356- 4C F3 D0      0026            JMP $D0F3  (<C343)  ;SUCHT A$(1)
0359- A0 02         0027 SPRUNG     LDY #$02          ;STRINGANFANGSADRESSE
035B- B1 44         0028 MARKE1     LDA (ADRESSE),Y     ;UND -LAENGE
035D- 99 4D 00      0029            STA PUFFER,Y        ;NACH PUFFER
0360- 88            0030            DEY                 ;SPEICHERN
0361- 10 F8         0031            BPL MARKE1
0363- A5 4D         0032            LDA *PUFFER         ;STRING LEER ?
0365- F0 1D         0033            BEQ ENDE            ;JA:=> ENDE
0367- C8            0034            INY
0368- B1 4E         0035 MARKE2     LDA (PUFFER+1),Y    ;STRINGINHALT NACH
036A- 99 7A 02      0036            STA $027A,Y       ;$027A FF. UEBERTRAGEN
036D- C8            0037            INY
036E- C4 4D         0038            CPY *PUFFER         ;FERTIG ?
0370- D0 F6         0039            BNE MARKE2          ;NEIN:=> WEITER
0372- A9 00         0040            LDA #$00            ;STRING MIT 0
0374- 99 7A 02      0041            STA $027A,Y         ;ABSCHLIESSEN.
0377- 20 2F E5      0042            JSR $E52F  (<E3B6)  ;ZEILENVORSCHUB
037A- A9 7A         0043            LDA #$7A            ;STRING AUF
037C- A0 02         0044            LDY #$02            ;BILDSCHIRM AUSGEBEN
037E- 20 1C CA      0045            JSR $CA1C  (<BB1D)  ;=AUSGABEROUTINE
0381- 20 2F E5      0046            JSR $E52F  (<E3B6)  ;ZEILENVORSCHUB
0384- 00            0047 ENDE       BRK
                    0048            .EN

LABEL FILE:  ( / = EXTERNAL )

/ANZAHL=0005          /DIMFLAG=0006          /NAME=0042
/ADRESSE=0044         /PUFFER=004D            ANFANG=033A
 SPRUNG=0359           MARKE1=035B            MARKE2=0368
ENDE=0384
```

```
                    0001            ;BEISPIEL ZU I.
                    0002            ;------------------------------
                    0003            ;
                    0004 INTADR     .DE $027A     ;ADRESSE DER INTEGERZAHL
                    0005 ADRESSE    .DE $44       ;VARIABLENADRESSE
                    0006            ;
                    0007            ;------------------------------
                    0008            .BA $033A
033A- 20 70 00      0009 ANFANG     JSR $0070         ;KOMMA UEBERLESEN
033D- 20 6D CF      0010            JSR $CF6D (<C12B);VARIABLENNAME AUS
                    0011            ;                  BASIC-TEXT HOLEN
                    0012            ;                  UND VARIABLE SUCHEN
0340- A0 01         0013            LDY #$01
0342- B9 7A 02      0014 MARKE      LDA INTADR,Y      ;INTEGERZAHL HOLEN
0345- 91 44         0015            STA (ADRESSE),Y   ;UND ALS VARIABLE
                    0016            ;                  B% ABLEGEN
0347- 88            0017            DEY
0348- 10 F8         0018            BPL MARKE         ;FERTIG ?
034A- 60            0019            RTS               ;JA:=>ZURUECK ZU BASIC
                    0020            .EN

LABEL FILE:   ( / = EXTERNAL )

/INTADR=027A            /ADRESSE=0044           ANFANG=033A
MARKE=0342
```

J. Verschieben eines Speicherblockes BLTUC 49887 < 45911

Wichtige Speicherzellen

Adresse	Bedeutung
92=$5C	Anfangsadresse des zu verschiebenden Blockes LB
93=$5D	Anfangsadresse des zu verschiebenden Blockes HB
87=$57	Endadresse+1 des zu verschiebenden Blockes LB
88=$58	Endadresse+1 des zu verschiebenden Blockes HB
85=$55	Endadresse+1 des Zielbereiches LB
86=$56	Endadresse+1 des Zielbereiches HB

Voraussetzungen

Die Grenzen des zu verschiebenden Blockes müssen in 92,93 bzw. 87,88 angegeben werden. Die Endadresse+1 des Zielbereiches muss in 85,86 stehen.

Ablauf der Routine

Die Routine errechnet zuerst aus der Länge des zu verschiebenden Speicherbereiches die Anzahl der 256-Byte-Abschnitte und die Länge des Restabschnittes
Mittels indirekt indizierter Adressierung wird dann in einer Verschiebeschleife der angegebene Speicherblock in den Zielbereich kopiert.

4. Ein/Ausgabe

Ohne die Möglichkeit der Ein/Ausgabe wäre ein sinnvoller Einsatz des Rechners unmöglich.
Die Steuerung von Geräten, über die der Rechner Daten ein- und ausgeben kann, ermöglicht überhaupt erst die Korrespondenz mit ihm.

Solche Geräte sind z.B. der Bildschirm und die Tastatur, die beiden Kassettenrekorder und jedes beliebige Gerät, das über den IEEE-Bus oder den USER-Port an den Rechner angeschlossen werden kann (Diskettenstation, Drucker, Analog-Digital-Wandler, Digital-Analog-Wandler, Steuerknüppel, usw.).

Diese Geräte müssen vom Rechner bedient werden, das heisst, der Rechner muss kontrollieren, ob die Geräte bereit sind, Daten aufzunehmen oder zu senden.

Ebenso muss gewährleistet sein, dass kein Peripheriegerät Daten sendet, währenddem der Rechner beispielsweise mit Berechnungen beschäftigt ist. Auch dürfen keine Daten vom Rechner zu einem gerade arbeitendem Gerät gesendet werden.

Während eines Datentransfers muss auch die Geschwindigkeit der Datenübertragung stimmen. Diese Geschwindigkeit wird als BAUDRATE bezeichnet (=Bit/Sekunde).
Das Nichtbeachten eines dieser Faktoren würde zu Datenverlusten oder gar zum Nichtfunktionieren der Rechner-Peripherie-Konfiguration führen!

Damit dies alles ermöglicht werden kann, besitzt der Rechner Anschlüsse, die nach bestimmten Normen mit den Peripheriegeräten korrespondieren, sogenannte SCHNITTSTELLEN.

4.1 Schnittstellen eines Rechners

Um mit peripheren Geräten arbeiten zu können, benötigt der Rechner einen Anschluss, die sogenannte Schnittstelle bzw. den Bus.
An dieser Schnittstelle werden die Daten-, Adress- und Steuerleitungen des Computers mit den Signalleitungen von Druckern, Floppys oder anderen Peripheriegeräten verbunden.

Hierbei haben sich vier Normen für Schnittstellen durchgesetzt:

a. Die V.24-Schnittstelle

b. Die RS-232-Schnittstelle

c. Der S-100-Bus

d. Die IEC-Bus-Schnittstelle

Diese Schnittstellen haben folgende Eigenschaften:

a. Die V.24-Schnittstelle

Sie besteht aus 25 Leitungen für die Uebertragung von Steuerzeichen und zur Synchronisation des Daten-Transfers. Sie bietet die Möglichkeit der Uebertragung von bis zu 20 KBit pro Sekunde über ca.15-20m Entfernung.

Diese Schnittstelle ist seriell, d.h. es wird ein Bit des Datenwortes nach dem anderen im ISO-7-Bit-Code auf der Sende- oder Empfangsleitung übertragen als Signale der Spannungspegel 3 bis 15 Volt (=LOW) und -3 bis -15 Volt (=HIGH).

Die anderen Leitungen dienen der Fehlererkennung und Einstellung der Uebertragungsgeschwindigkeit.
Ihren Namen hat sie als Empfehlung V.24 und V.28 von der CCITT (Comite Consultatif International Telegraphique et Telephonique)
Sie wird über den 25-poligen V.24-Stecker nach DIN 66 020 angeschlossen.

Mit den 25 Leitungen wurden alle möglichen Uebertragungsfälle abgedeckt, wogegen in der Praxis jeweils nur eine Auswahl genutzt wird.
Die Auswahl der Steuerleitungen reduziert sich im Extremfall auf die Steuerleitungen der sehr gebräuchlichen RS 232-C-Schnittstelle.

b. RS 232-Schnittstelle (genormt nach EIA RS 232C)

Hier sind lediglich noch 3 Signalleitungen plus Schutzerde vorgesehen.

Es handelt sich hierbei um eine serielle 20mA-Current-Loop-Schnittstelle, die mit 75-19200 Baud betrieben wird.
Sie erlaubt den Anschluss mehrerer Geräte auch bei Entfernungen von mehr als 1 km und verfügt über keinerlei Steuerleitungen.
Die Fehlermöglichkeiten werden durch entsprechende, festeingestellte Uebertragungsraten abgefangen.

Durch den vielseitigen Einsatz bei Teletypes, Datenterminals und Druckern sowie durch die grosse Uebertragungsentfernung hat sie sich in der EDV gegenüber den übrigen 3 Schnittstellen am stärksten durchgesetzt.

Verwendet wird der 25-polige Stecker nach V.24 (genormt nach DIN 66 020), von dem allerdings nur die Stifte
1 (Schutzerde), 2 (Ausgang), 3 (Eingang) und
7 (Betriebserde) belegt werden.
Die Schutzerde (Pin 1) wird bei Mikrocomputeranwendungen nicht genutzt.

c. Der S-100-Bus

S-100 steht für "STANDARDBUS mit 100 Leitungen" .
Im Gegensatz zu den bisher besprochenen Schnittstellen handelt es sich hierbei nicht um einen Peripheriebus, sondern im Grunde um einen Systembus zur Realisierung von beispielsweise verschiedenen Aus- und Eingabemöglichkeiten (V.24-, RS 232-, IEC-Schnittstellen) sowie zur Erweiterung des Rechnersystems durch Floppy-Disk-Control-Bausteine, Speichererweiterungen, AD- und DA-Wandler usw.
Auf diesem Bus liegen alle Steuersignale des Prozessors wie z.B. Interruptanforderung, Direct Memory Access, Taktsignale und Betriebsspannungen.
Der S-100-Bus ist nicht mehr seriell organisiert, sondern benutzt je 8 Leitungen als Datenein- und Ausgänge sowie je 16 Leitungen als Adressein- und ausgänge.
In der Praxis wird auch hier - wie bei der V.24-Schnittstelle - nur eine Auswahl von Leitungen benutzt.

Der S-100-Bus als Systembus hat sich in der Industrie bereits weitgehend durchgesetzt: Für den Aufbau eines modularen Computersystems sind im Handel derzeit nahezu alle Bausteine erhältlich.

d. Der IEC-Bus entstand aus der Absicht heraus, die Vorteile des S-100-Systembusses (mehrere Geräte gleichzeitig anschliessbar, Daten parallel verfügbar) mit den Eigenschaften der V.24-Schnittstelle (grössere Uebertragungsstrecke) zu kombinieren.

Diese Möglichkeiten erweisen sich vor allem in der Messtechnik als vorteilhaft, wenn es beispielsweise um den Anschluss mehrerer Mess- und Steuergeräte an den Computer geht.

Der IEC-Bus wurde Anfang der siebziger Jahre von der Firma Hewlett Packard (HP) mit dem HP-Interface-Bus erstmals realisiert.
Dieser Bus besteht aus 16 Leitungen, die sich wie folgt aufteilen:

8 Datenleitungen zur 8-bit-parallelen Datenübertragung, an die bis zu 15 Geräte angeschlossen werden können,

5 sogenannte Management-Signalleitungen und

3 Kontrolleitungen.

Dieser Firmenvorschlag von Hewlett Packard wurde von den internationalen Normengremien übernommen:

IEEE (Institute of Electronic and Electrical Engineers) - Standard IEEE 488/1975

IEC (International Electrotechnical Commission) - IEC 66.22

DIN-IEC (Deutsches Institut für Normung) - DIN IEC 66.22

Die Normen stimmen in Bezug auf die Funktionen des Busses überein, allerdings unterscheiden sie sich in der Verwendung der Stecker:

a. 25-poliger V.24-Stecker nach DIN 66020
b. 24-poliger Hewlett Packard-Stecker nach IEEE.

Konfiguration des IEC-Busses

An den IEC-Bus sind folgende Gerätetypen anschliessbar:

Talker: (=Sprecher), -Geräte, die nur als Datenquelle (Sender) arbeiten.

Listener: (=Hörer), -Geräte, die nur als Datenempfänger arbeiten.

Talk&Listen: Geräte, die Daten sowohl empfangen als auch senden können.

Controller: Gerät, das als Kontrollgerät wirkt: Der Rechner.

Es darf immer nur EIN Controller am IEC-Bus angeschlossen sein. Ausserdem darf immer nur jeweils EIN Talker arbeiten!

Bedeutungen der Kontrolleitungen

a. 3 Handshake Leitungen:

DAV	- Data Valid	(Daten gültig)
NRFD	- Not Ready For Data	(Nicht aufnahmebereit für Daten)
NDAC	- Not Data Accepted	(keine Daten angenommen)

b. 5 Management Leitungen:

ATN	- Attention	(Achtung)
IFC	- Interface Clear	(Interface bereit)
SRQ	- Service Request	(Service-Anforderung)
REN	- Remote Enable	(Fernbedienung möglich)
EOI	- End Or Identify	(Ende oder Kennung)

Besonderheiten des IEC-Busses

Die Datenübertragungsgeschwindigkeit ist festgelegt auf maximal 1 Mbyte/s (1 Million Byte pro Sekunde).

Die gesamte Kabellänge (=Buslänge) darf 20m nicht überschreiten: beim Anschliessen aller möglichen 15 Teilnehmer darf die Kabellänge pro Gerät max. 1.50 m betragen.

Die Geräte sollten möglichst in Reihe aneinandergesteckt werden (keine Sternpunktverkabelung!).

Der CBM verwendet ebenfalls einen IEC-Bus, dessen Signale allerdings etwas von der Norm abweichen.

4.2 Der CBM-IEC-Bus

Zur Kommunikation mit der Aussenwelt besitzt der CBM neben dem parallelen USER-PORT (siehe 4.5) auch den in diesem Abschnitt beschriebenen IEC-Bus, bei dem die Anschlussbelegung der IEEE-488-Norm entspricht (siehe Anhang 25).

Die Signale entsprechen allerdings leider nicht exakt der Norm, so dass beim Anschluss von manchen IEC-Bus Geräten Schwierigkeiten auftauchen werden.

Bei den serienmässigen Commodore-Geräten treten diese Schwierigkeiten natürlich nicht auf.

4.2.1 Ablauf des Handshakes beim CBM-IEC-Bus

a. Im Ruhezustand ist die NRFD-Leitung auf 'not ready', DAV auf 'data not valid' und NDAC auf 'data not accepted'. Wird nun zum Beispiel das Peripheriegerät fertig, wenn es als LISTENER arbeitet und der CBM als TALKER, so zeigt es dies durch "Hochnehmen" der NRFD-Leitung auf 'ready for data' an.
   Ist dagegen der CBM gerade LISTENER und das Gerät TALKER (z.B. nach einem INPUT#-Befehl), so

b. nimmt der CBM die Leitung NRFD auf 'ready for data' und die Daten müssen spätestens 64 ms danach mit DAV als gültig angezeigt werden, da sonst eine Fehlermeldung im CBM-Status erscheint.

c. DAV geht nun auf 'data valid' und zeigt damit an, dass die Daten auf dem Datenbus verwendet werden können.

d. Vom LISTENER muss nun als erstes die NRFD-Leitung wieder auf 'not ready for data' genommen werden und dann

e. die Leitung NDAC auf 'data accepted', um dem TALKER anzuzeigen, dass die Daten aufgenommen wurden.

f. Dieser reagiert dann durch DAV auf 'data not valid' und nimmt die Daten vom Datenbus weg, sobald

g. der LISTENER NDAC wieder auf 'data not accepted' gebracht hat.

Eine Zeitüberwachung von seiten des CBM liegt auch vor, wenn er als Talker DAV auf 'data valid' schaltet und NDAC abwartet; NDAC muss innerhalb von 64 ms auf 'data accepted' gehen.

Diese Zeitüberwachung macht insbesondere bei langsamen Speichern Schwierigkeiten, die über den IEC-Bus angeschlossen werden sollen. Wird ein Richtungswechsel beim CBM durchgeführt, also vom TALKER auf LISTENER (ggf. durch einen INPUT#- oder GET#-Befehl), so müssen die Daten spätestens nach 64 ms anstehen, da es nicht möglich ist, das Handshake bis zur Aufnahme der Daten vorher anzuhalten. Ist einmal das erste Byte gegeben, so kann das Handshake zum Beispiel durch Belassen des DAV im Zustand 'data valid' beliebig lange gehalten werden. Probleme gibt es also nur beim ersten Byte. Gerade bei Floppys oder Platten muss dieses Byte aber unter Umständen erst geholt werden, und dafür ist die Zeit viel grösser als 64 ms! Bei dem INPUT#-Befehl ergeben sich keine Schwierigkeiten, da dem CBM beliebig viele Blanks gesendet werden können, die dieser dann ignoriert; nur bei GET# ist dies nicht möglich, und Abhilfe lässt sich nur durch ein 'vorbeugendes' Holen des Datensatzes vor der Ausführung des GET#-Befehles schaffen.

4.2.2 Die IEC-Management-Leitungen

ATN Attention wird vom CBM auf LOW gesetzt, wenn er Kommandos über den Bus schickt.
Wenn ATN auf LOW ist, sind also immer nur Kommandos oder Peripherie-Adressen auf dem Datenbus.
Ist ATN auf HIGH-Pegel, so können nur zuvor adressierte Geräte auf dem Bus Daten austauschen.

EOI End or Identify. Wird das letzte Byte eines Datensatzes übertragen, so kann EOI auf LOW gesetzt werden. Der CBM setzt EOI immer beim letzten Byte auf LOW, um das Ende des Datenstromes anzuzeigen. Ein angeschlossenes Gerät muss EOI setzen, wenn es solche Daten zum CBM überträgt (deren letztes Zeichen nicht CR ist), um das Ende der Uebertragung anzuzeigen.

IFC Interface clear. Nach einem RESET oder beim Einschalten des CBM wird dieses Signal für ungefähr 100 ms auf LOW gelegt.

SRQ Service Request. Wird vom BASIC nicht benützt, ist aber durch POKE setzbar.

REN Remote enable. REN wird beim CBM permanent auf LOW gelegt.

4.2.3 Statusmeldungen des CBM

Ueber die Variable ST, die als "Nur-Lese-Speicher" fungiert, kann vom BASIC aus der Status des IEC-Bus-Transfers abgelesen werden:

ST=1	bedeutet Zeitüberlauf beim LISTENER; es wurde nach 64 ms keine Reaktion gezeigt.
ST=2	bedeutet Zeitüberlauf beim TALKER. DAV zeigt nach 64 ms seit 'ready for data' keine Antwort.
ST=64	EOI ging beim letzten Datentransfer auf LOW.
ST=-128	DEVICE NOT PRESENT. Hier verlässt BASIC die Ausführungsphase und geht in den BASIC-Kommando-Zustand mit Ausgabe der Error-Meldung.

In Maschinensprache kann der Status des IEC-Bus durch die Abfrage der Speicherstelle 150=$96 eingesehen werden. Die enthaltenen Werte entsprechen denen der BASIC-Variablen ST.

Im BASIC 4 wurde die Kontrolle der ordnungsgemäßen Befehlsdurchführung und Fehlerprüfung beim Arbeiten mit der Diskettenstation durch die Einführung von 2 weiteren Statusvariablen, nämlich ds und ds$, wesentlich komfortabler.
Dabei enthält beim Auftreten eines Fehlers beim Diskettenhandling die Variable ds den Diskstatus und ds$ die Art des Fehlers in Klartext. Die Bedeutung der verschiedenen Aussagen dieser Variablen entnehmen Sie bitte dem BASIC 4-Handbuch.

4.3 Die Konversation mit dem IEC-Bus

Der IEC-Bus verlangt eine bestimmte Sequenz von Operationen, damit ein reibungsloser Datentransfer über ihn möglich wird.

Um dies zu veranschaulichen, soll der Ablauf eines IEC-Bus-Transfers anhand von BASIC-Befehlsfolgen aufgezeigt werden:

Zu Anfang muss ein logisches File (Nr.1-255, beliebig) für ein bestimmtes IEC-Bus-Gerät (=physikalische Adresse) mit Angabe dessen Funktion als LISTENER oder TALKER und evtl. dem Namen des Files eröffnet werden.
Erst dann können Daten übertragen werden!

Nach Beendigung der Datenübertragung muss als Abschluss dem Gerät ein UNLISTEN oder UNTALK gesendet werden.

Dies entspricht insgesamt beispielsweise der BASIC-Syntax:

a. OPEN 1,8,4,"INPUT"

Dabei wird die logische File-Nr.1 (=Primäradresse) der physikalischen Adresse 8 zugeordnet (=Floppy-Disk) mit der Sekundäradresse=4.
Als Name des Files wird die Zeichenfolge "INPUT" übertragen.

Auf dem IEC-Bus sieht das folgendermassen aus:
Da der CBM Kommandos senden will, setzt er ATN=LOW.

Es folgen die Kommandobytes:
$28 (2:Listener/8:Device-Adresse)
$F4 (F:OPEN-Befehl/4:Sekundäradresse)
Nach ATN=HIGH folgt der Transfer der Daten:
$49 = 'I'
$4E = 'N'
$50 = 'P'
$55 = 'U'
$54 = 'T' mit gleichzeitigem EOI=LOW
Sodann wird bei ATN=LOW der UNLISTEN-Befehl an das Gerät übermittelt:
$3F (UNLISTEN-Befehl)

Ähnlich verläuft auch die Signalfolge bei der BASIC-Befehlssequenz

b. PRINT#1,"TEST"

(Vorher muss OPEN 1,8,4 erfolgt sein!)

Die Signale sehen dann wie folgt aus:

```
   ATN=LOW  $28   (2:LISTENER/8:Device-Nr.)
            $64   (6:PRINT-Befehl/4:Sek.-Adr.)
   ATN=HIGH $54   'T'
            $45   'E'
            $53   'S'
            $54   'T'  mit gleichzeitigem EOI=LOW
   ATN=LOW  $3F   (UNLISTEN-Kommando)
```

c. INPUT#1,A$ (Eingabe eines Strings)

```
   ATN=LOW  $48   (4:TALKER/8:Dev.-Nr.)
            $64   (6:INPUT-Befehl/4:Sek.Adr.)

            Es erfolgt die Eingabe vom eingestellten
            Eingabegerät:
     z.B.:  $41   'A'
            $42   'B'    (="ABC12")
            $43   'C'
            $31   '1'
            $32   '2'  mit EOI=LOW
   ATN=LOW  $5F   (UNTALK-Befehl)

            A$ enthält nun den String "ABC12".
```

Wenn der Datentransfer abgeschlossen ist und das Gerät zeitweilig nicht mehr gebraucht wird, so muss das dazugehörige File geschlossen werden:

d. CLOSE1

```
   ATN=LOW  $28   (2:LISTENER/8:Dev.-Adr.)
            $E4   (E:CLOSE-Befehl/4:Sek.-Adr.)
            $3F   (UNLISTEN-Befehl)
```

e. LOAD und SAVE

Soll der LOAD-Befehl ausgeführt werden, so wird der Sekundärkanal 0 verwendet, für den SAVE-Befehl der Sekundärkanal 1.

Bei LOAD"Name",Dev.Adr. wird zunächst ein
OPEN 1,Dev.Adr.,0,"Name" und dann ein
INPUT#1,Speicher ausgeführt.
Den Abschluss bildet CLOSE1

Bei SAVE"Name",Dev.Adr. wird analog verfahren, nur wird anstatt der Sekundäradresse 0 die 1 verwendet und es erfolgt ein PRINT#1,Speicherbereich

4.4 IEC-Bus-Routinen

4.4.1 IEC-Bus-Steuerung in Maschinensprache

Die Reihenfolge der IEC-Bus-Kommandos muss zur Vermeidung von Fehlern auch bei der Programmierung in Maschinensprache unbedingt beibehalten werden !

Der IEC-Bus verlangt eine spezielle, zeitabhängige Reihenfolge von Signalen auf den Steuer- und Managementleitungen.
Diese Signale sind schon in der allgemeinen Besprechung über die Bedeutungen der Steuer- und Managementleitungen (Kapitel 4.2.1) einführend behandelt worden. Detaillierte Angaben über Timing und Signalfolge können in weiterführender Literatur nachgelesen werden und würden hier nur den Rahmen sprengen.
Bei Interesse an den Signalverläufen dieser Leitungen wird auch das Studium des ROM-Listings der im CBM-Betriebssystem implementierten IEC-Bus-Routinen weiterhelfen. Diese Routinen behandeln die Leitungen des IEC-Bus und werden vom Betriebssystem zur Ausführung eines Datentransfers verwendet.

Die IEC-Bus-Routinen können dem Assemblerprogrammierer ebenfalls die Aus- und Eingabe über den IEC-Bus erleichtern und können von ihm auch ohne Kenntnisse über die Signalverläufe verwendet werden.

Die Reihenfolge ist dabei immer die folgende:

a. File eröffnen

b. evtl. Bus testen (nicht unbedingt nötig)

c. Datentransfer

d. File schliessen

e. IEC-Bus-Adressen rückstellen

Diese Reihenfolge ist zwingend !

Bevor ein Datentransfer über den IEC-Bus erfolgen kann, muss vom Controller (beim CBM immer der Rechner) mindestens ein Peripheriegerät zum TALKER oder LISTENER erklärt worden sein.

Dabei dürfen gleichzeitig nur ein TALKER, jedoch meh- rere LISTENER angesprochen sein.
Es können insbesondere auch Datentransfers direkt unter den Peripheriegeräten stattfinden ohne Eingreifen des Controllers.

Ist zwischen einer bestimmten Gerätekonfiguration der Datenaustausch beendet, so müssen sie mit dem Befehl UNLISTEN bzw. UNTALK in ihre Grundstellung gebracht werden, um so auf neue Kommandos zu warten, ohne am Datentransfer teilzunehmen.

Um diese TALKER oder LISTENER mit Daten zu versorgen, müssen sogenannte "Files" eröffnet werden, die quasi Kanäle zu den einzelnen Geräten bilden.

Vom Rechner wird danach nur in diese Files geschrieben, bzw. aus ihnen gelesen.
Für ihn sind also die FILES so etwas wie Peripheriegeräte.

Nach Benutzung sind die Files wieder zu schliessen.

Den Files sind bestimmte Parameter zugeordnet:

```
OPEN a,b,c,"Name"
```

a. Die logische File-Nummer: Nach Eröffnen des Files wird dieses nur noch über seine logische Nummer angesprochen, die zwischen 0 und 255 liegen kann.

b. Die Primäradresse : Sie gibt die physikalische Adresse des Gerätes (Gerätenummer) an, mit dem das File arbeiten soll. Die Adresse des Gerätes kann zwischen 0 und 15 liegen und ist hardwaremässig festgelegt.

c. Die Sekundäradresse: Es gibt Geräte, die selbst mehrere Funktionen ausführen können, etwa verschiedene Betriebszustände (z.B. Drucker) oder verschiedene Datenkanäle (z.B. Floppy). Solche Geräte benötigen deshalb eine sogenannte Sekundäradresse, durch die die Betriebsart bzw. der Datenkanal festgelegt wird.
Man kann dadurch beispielsweise Fehlermeldungen ermöglichen oder ein bestimmtes Ausgabeformat erreichen.
Den Begriff Sekundäradresse verwendet man deshalb, weil die Gerätenummer auch Primäradresse genannt wird.
Die Sekundäradresse kann zwischen 0 und 31 liegen.

Gleichzeitig kann nur eine Datei mit einer bestimmten logischen Adresse geoeffnet sein.
Wird versucht, zum zweiten Mal ein OPEN mit der gleichen logischen Adresse durchzuführen, ist die Folge ein "FILE OPEN ERROR".
Vom Betriebssystem sind maximal 10 Dateien gleichzeitig zugelassen. Wird diese Anzahl überschritten, erfolgt die Fehlermeldung "TOO MANY FILES".
Bei beiden Fehlermeldungen werden gleichzeitig alle anderen Dateien vom Betriebssystem geschlossen.
Die Zuordnung zwischen den einzelnen logischen Adressen, Primär- und Sekundäradressen wird vom Betriebssystem in einer Tabelle verwaltet (593-622).

Diese Tabelle gliedert sich wie folgt:

593-602	Liste der File-Nummern
603-612	Liste der Primäradressen
613-622	Liste der Sekundäradressen

Die Anzahl der offenen Files steht in 174. Falls nun eine der beiden Fehlermeldungen "FILE (NOT) OPEN ERROR" auftritt, durch die ja alle Dateien geschlossen werden, kann man den Zähler in 174 auf die Anzahl der vorher offenen Dateien setzen und dadurch diese wieder öffen.
Dies ist deshalb möglich, weil der Inhalt der eigentlichen Tabelle durch diese Fehlermeldungen nicht verändert wird, sondern nur durch CLOSE.

Das Verständnis dieser Bedeutungen ist für die folgenden Kapitel zur IEC-Bus-Behandlung von grösster Wichtigkeit.

Damit die verschiedenen Kontrollfunktionen auf dem IEC-Bus ausgeführt werden können, sind im Betriebssystem des CBM die verschiedensten Routinen implementiert, die im folgenden beschrieben werden sollen.

A. OPEN IEC-Bus — 62756 < 62819

Wichtige Speicherzellen

Adresse	Bedeutung
209=$D1	Länge des Filenamens
210=$D2	Logische File-Nummer
211=$D3	Sekundäradresse
212=$D4	Gerätenummer=Primäradresse
218,219=$DA,DB	Zeiger auf den Anfang des Filenamens

Voraussetzung

Alle oben angegebenen Speicherzellen müssen auf die gewünschten Werte gesetzt werden.

Ablauf der Routine

Diese Routine entspricht dem BASIC-Befehl OPEN a,b,c,"Name": Sie eröffnet eine Datei, die erst nach Aufruf dieser Routine angesprochen werden kann.

Die Parameter des File werden in der Filetabelle abgelegt und können mit späteren Routinen wieder abgerufen werden.

Diese Routine bildet die notwendige Voraussetzung für alle anderen IEC-Bus-Routinen !

B. CLOSE IEC-Bus — 62124 < 62176

Wichtige Speicherzellen:

Adresse	Bedeutung
210=$D2	Logische Filenummer des zu schliessenden Files

Voraussetzungen:

Das File muss vorher eröffnet worden sein. (dadurch werden die File-Parameter in die Filetabelle eingetragen).

Ablauf der Routine:

Die Routine sucht zunächst in der Filetabelle nach der Nummer des zu schliessenden Files und kehrt bei Nichtauffinden in das aufrufende Programm zurück.
Ist das File vorhanden, so wird es geschlossen, indem in der Filetabelle seine Parameter durch die Parameter des zuletzt geöffneten Files ersetzt werden.

Dabei berücksichtigt die Routine folgenden Spezialfall:
Wenn das File zum Schreiben auf einen Kassettenrekorder geöffnet war, so wird zunächst der letzte Block geschrieben.
Wenn die Sekundäradresse ($D3) den Wert $62 hat, so folgt ein Bandende-Block. Erst dann kehrt die Routine in das aufrufende Programm zurück.

C. File-Nummer suchen — 62093 < 62145

Will man aus einem Assemblerprogramm mit mehreren Dateien arbeiten (z.B. : Eingabe, Ausgabe, Kommando- und Fehlerkanal der Floppy-Disk), so ist es etwas umständlich, immer für jedes File die Angaben logische Filenummer, Primär- und Sekundäradresse für die I/O-Routine zur Verfügung zu stellen. Diese Angaben sind im Rechner in Tabellen enthalten und können von dort mit dieser und der folgenden Routine abgerufen werden.

Voraussetzungen

Die logische Nummer des zu suchenden Files muss im AC übergeben werden.

Ablauf der Routine

Wenn vorher ein File mit der zu suchenden Nummer eröffnet worden ist, so erhält das XR den Stellenwert der Filenummer in der Filetabelle, danach erfolgt Rücksprung in das aufrufende Programm.
War kein File mit der logischen Nummer=(AC) geöffnet, so hat das XR nach der Rückkehr den Wert =0.

D. File-Parameter setzen — 62105 < 62157

Voraussetzungen

Im XR muss die Stelle in der Tabelle angegeben werden, an der die logische Nummer des Files steht, welche als laufendes File gelten soll.

Das File muss dafür vorher geöffnet worden sein !

Ablauf der Routine

Das File, dessen Parameter in der Filetabelle an der Stelle stehen, die im XR angegeben ist, wird als laufendes File gesetzt.

Am einfachsten wird diese Routine in Verbindung mit der Routine C. verwendet:
Zunächst wird in der Tabelle nachgeschaut, ob das File schon geöffnet wurde (C.). Das XR enthält nun den Index für die Stelle der Fileparameter in der Filetabelle. Mit diesem Index kann dann die Routine D. ausgeführt werden:
Es werden die Parameter des bestimmten Files (Primär-, Sekundär-adresse und Filenummer) als Parameter des laufenden Files eingesetzt.

Die Verwendung der beiden soeben beschriebenen Routinen (C. und D.) ermöglicht eine schnellere und komfortablere Verwaltung von mehreren Dateien (wie z.B Ausgabe auf Drucker UND Ausgabe auf Bildschirm UND Eingabe vom Kassettenrekorder), ohne die umständliche Bereitstellung der Fileparameter der OPEN IEC-Bus-Routine berücksichtigen zu müssen.

Allerdings muss das File im Programm vorher einmal geöffnet worden sein !

Beispiel: Ein bestimmtes File soll als laufendes I/O-File benannt werden:

```
LDA #NUMMER   ;FILENUMMER LADEN
JSR SUCHEN    ;SUCHT IN TABELLE (ROUT.C)
BNE FEHLER    ;NICHT GEFUNDEN:=>FEHLER
JSR SETZEN    ;ALS LAUFENDES FILE SETZEN
               =ROUTINE D.
```

E. TALKER-Kommando senden — 61622 < 62685

Voraussetzungen

Zu dem Gerät, das als TALKER bestimmt werden soll, muss ein File geoeffnet und als laufendes File bestimmt worden sein.

Ablauf der Routine

Das Gerät, das vom laufenden File angesprochen wird, wird zum TALKER.

F. LISTENER-Kommando senden	LISTN	61626 < 61653

Voraussetzungen

Zu dem Gerät, das als LISTENER bestimmt werden soll, muss ein File bestimmt worden sein.

Ablauf der Routine

Das Gerät, das vom laufenden File angesprochen wird, wird zum LISTENER.

G. UNTALK-Kommando senden	UNTLK	61823 < 61870

Ablauf der Routine

Sämtliche TALKER am IEC-Bus werden auf ihre Grundstellung gebracht, in der sie auf Kommandos warten und währenddessen nicht am Datentransfer teilnehmen.

H. UNLISTEN-Kommando senden	UNLSN	61827 < 61881

Ablauf der Routine

Sämtliche LISTENER am IEC-Bus werden auf ihre Grundstellung gebracht, in der sie auf Kommandos warten und währenddessen nicht am Datentransfer teilnehmen.

I. Normale Ein/Ausgabe setzen	CLRCH	62066 < 62118

Ablauf der Routine

Diese Routine setzt den IEC-Bus auf Eingabegerät=Tastatur und Ausgabegerät=Bildschirm, also die Konfiguration, die nach Einschalten des Rechners besteht.
Die geöffneten Dateien sind damit nicht wirklich geschlossen. Ein Schliessen erfolgt erst mit CLOSE (siehe B.).

4.4.2 Ausgabe über den IEC-Bus

Im Grunde sind alle Ausgabemöglichkeiten des CBM (ausser User-Port) über den IEC-Bus realisiert: Die Routinen, die für den Bildschirm gedacht sind, können sehr einfach durch einen Befehl (BASIC:CMD) auf ein beliebiges Gerät umgelenkt werden.

Dabei nehmen die beiden Kassettenrekorder eine besondere Stellung ein:
Auch sie können als intelligente "IEC-Bus-Geräte" angesprochen werden. Allerdings ist ihre Intelligenz durch Hardware auf der Rechnerplatine und spezielle Treibersoftware im ROM realisiert. Ihr Verhalten ähnelt jedoch (mit einigen Ausnahmen) dem eines üblichen IEC-Bus-Gerätes.

Sobald also eines der speziellen Geräte, wie Bildschirm, Tastatur oder Kassettenrekorder angespochen wird, verzweigen die beschriebenen Routinen in Unterprogramme, die das gleiche bei diesen Geräten bewirken, wie sie es bei den regulären IEC-Bus Geräten tun würden.

A. Allg. Ausgabe auf den IEC-Bus	CKOUT	63420 < 63486

Voraussetzungen

Das File, das angesprochen werden soll, muss geöffnet worden sein (OPEN IEC-Bus) !

Das X-Register muss die logische Filenummer enthalten.

Ablauf der Routine

Die Routine bewirkt, dass alle Ausgaben, die normalerweise auf dem Bildschirm erscheinen würden, auf das Gerät ausgegeben werden, welches über das File angesprochen wird.

Diese Routine entspricht dem BASIC-Befehl CMD(Nr.).

Damit sind auch Möglichkeiten zur Bildschirmausgabe (siehe 4.4.3) nach Aufruf dieser Routine (soweit sinnvoll) für ein beliebiges IEC-Bus-Gerät gültig !

```
LDX FILENUMMER
JSR OUTPUTCMD
```

Diese Einstellung bleibt solange bestehen, bis entweder mit der gleichen Routine ein anderes File adressiert wird, oder mittels Routine I. (siehe unter 4.4.1) wieder auf normale Ein/Ausgabe geschaltet wird.

```
                    0001            ;BEISPIEL ZU A: ANSTEUERUNG
                    0002            ;EINES DRUCKERS AM IEC-BUS:
                    0003            ;----------------------------------
                    0004 LOGNR      .DE $D2       ;LOG. FILENUMMER
                    0005 DEVNR      .DE $D4       ;GERAETENUMMER
                    0006 IECOPEN    .DE $F524     (< F563)
                    0007 IECTEST    .DE $F12D     (< F148)
                    0008 CMD        .DE $F7BC     (< F7FE)
                    0009 RESTORE    .DE $F272     (< F2A6)
                    0010            ;----------------------------------
                    0011            .BA $033A
                    0012            ;ENTSPRICHT DEM BASIC-BEFEHL:
                    0013            ;
                    0014            ;       OPEN4,4:CMD4
                    0015            ;
033A- A2 04         0016 START      LDX #$04
033C- 86 D2         0017            STX *LOGNR    ;FILE NR.=4
033E- 86 D4         0018            STX *DEVNR    ;AUSGABE:=>DRUCKER
0340- 20 24 F5      0019            JSR IECOPEN   ;OUTPUT-FILE EROEFFNEN
0343- 20 2D F1      0020            JSR IECTEST   ;IEC-BUS PRUEFEN
0346- A2 04         0021            LDX #04       ;XR=FILENUMMER
0348- 20 BC F7      0022            JSR CMD
034B- 60            0023            RTS
                    0024            ;ENTSPRICHT DEM BASIC-BEFEHL:
                    0025            ;
                    0026            ;       PRINT#4:CLOSE
                    0027            ;
034C- 4C 72 F2      0028 STOP       JMP RESTORE   ;EINGABE=TASTATUR,
                    0029            ;              AUSGABE=BILDSCHIRM
034F- 60            0030            RTS
                    0031            .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/LOGNR=00D2                 /DEVNR=00D4          /IECOPEN=F524 (< F583)
/IECTEST=F12D (< F148)      /CMD=F7BC (< F7FE)   /RESTORE=F272 (< F2A6)
 START=033A                  STOP=034C
```

Nach Aufruf dieses Programmes wird jede Ausgabe auf den Drucker umgeleitet, die sonst auf dem Bildschirm erscheinen würde. Die Rückstellung geschieht durch die Routine 62066=$F272 und ist im dargestellten Programm unter STOP realisiert (Einsprung mit SYS 844).

B. Sekundäradresse senden	SECND	61736 < 61763

Voraussetzungen

Es muss ein File eröffnet worden sein, für das die Sekundäradresse gesendet werden soll.

Die Sekundäradresse wird dazu im AC übergeben.

Ablauf der Routine

Die Routine sendet den Wert im AC im Kommandomodus als Sekundäradresse an das IEC-Bus-Gerät, das vom laufenden File angesprochen wird.

C. Ausgabe eines Zeichens zum IEC-Bus	CIOUT	61807 < 61854

Wichtige Speicherzellen

Adresse	Bedeutung
210=$D2	Logische Filenummer
211=$D3	Sekundäradresse
212=$D4	Primäradresse

Voraussetzungen

Das File, welches zur Ausgabe angesprochen werden soll, muss vorher geöffnet worden sein (OPEN IEC-Bus) !
Die oben angegebenen Speicherzellen müssen die aktuellen Fileparameter enthalten.

Das auszugebende Zeichen wird im AC übergeben.

Ablauf der Routine

Diese Routine ermöglicht die gesamte Kontrolle der Dateien vom Assemblerprogramm aus. Das Statusbyte ist nach der Datenübermittlung auf Fehler (<>0) zu überprüfen.

Es wird die Speicherzelle ($B0) abgefragt, auf welches Gerät der Akkumulator-Inhalt ausgegeben werden soll.

1/2: Cassettenrekorder
3 : Bildschirm
>3 : IEC-Bus-Gerät

1/2: Die Rekorder-Nr. muss in ($D4) abgelegt sein!
Das Zeichen aus dem Akkumulator wird in den Puffer ($027A-$0339 und $033A-$03F9) des angesprochenen Kassettenrekorders (1/2) geladen und der Pufferzähler ($BB,$BC) inkrementiert (=JMP 63494 < 63563 : Zähler erhöhen und nach YR bringen).
Wenn der Puffer voll ist (192 Bytes), so wird dieser als ein Block auf Band geschrieben.

3: Der Akkumulator-Inhalt wird auf den Bildschirm ausgegeben (=JMP 58328 < 57858). Die Vorbereitungsroutinen zur Einstellung des IEC-Bus können in diesem speziellen Fall umgangen werden durch Ausführung der Subroutine RESTORE-IEC-Bus (=$FFCC).
(Diese Routine stellt den IEC-Bus ein auf Tastatur als Eingabe- und Bildschirm als Ausgabegerät.)

>3: Der Akkumulatorinhalt wird auf das eingestellte IEC-Bus-Gerät ausgegeben (=JMP 61807 61854).

Um die Wirkung dieser etwas komplizierteren Ausgabe-möglichkeit zu erläutern, folgt das Beispiel auf der nächsten Seite.

Bevor das Beispielprogramm aufgerufen werden kann, muss ein File für das Gerät geöffnet worden sein, auf das ausgegeben werden soll. Dies kann erfolgen durch Aufruf der Routine OPEN IEC-Bus oder durch den BASIC-Befehl OPEN 1,?.

Der Einsprung in das Programm erfolgt durch JSR $033A bzw. SYS 826.

Jeder Aufruf des Programmes gibt dann das Zeichen "A" auf das IEC-Bus-Gerät aus.

```
                    0001 STATUS      .DE $96
                    0002 LISTEN      .DE $FOBA (<FOD5)
                    0003 UNLISTEN    .DE $F183 (<F1B9)
                    0004 SENDADR     .DE $F128 (<F143)
                    0005 AUSGABE     .DE $F16F (<F19E)
                    0006             ;---------------------------------
                    0007             .BA $033A
033A- A9 00         0008 ANFANG      LDA #$00      ;STATUS-WORT
033C- 85 96         0009             STA *STATUS   ;LOESCHEN
033E- 20 BA FO      0010             JSR LISTEN    ;GERAET = LISTENER
0341- A9 00         0011             LDA #$00      ;SEKUNDAERADRESSE
0343- 20 28 F1      0012             JSR SENDADR   ;AN IEC-BUS SENDEN
0346- A9 41         0013             LDA #'A       ;ZEICHEN
0348- 20 6F F1      0014             JSR AUSGABE   ;AUSGEBEN
034B- 20 83 F1      0015             JSR UNLISTEN ;GERAET=UNLISTEN
034E- A5 96         0016             LDA *STATUS   ;STATUS-WORT
0350- DO 00         0017             BNE FEHLER    ;UEBERPRUEFEN
                    0018 FEHLER      ;FEHLER BEI DER UEBERTRAGUNG
                    0019             ;BEHANDELN
                    0020             .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/STATUS=0096              /LISTEN=FOBA (<FOd5)     /UNLISTEN=F183 (<F1B9)
/SENDADR=F128 (<F143)     /AUSGABE=F16F(<F19E)      ANFANG=033A
FEHLER=0352
```

Durch mehrmalige Ausführung der Zeilen 13 und 14 kann die Ausgabe von ganzen Zeichenketten erreicht werden, ohne wegen jedem Zeichen die Kontrollfunktionen ausführen zu müssen.

In diesem Beispiel wird als Ausgabegerät der Drucker angesprochen: Bei jedem Aufruf des Programmes wird ein "A" in dessen Puffer geschrieben.

Auf dem Blatt erscheint es allerdings erst dann, wenn ein CARRIAGE RETURN(=$0D) oder der BASIC-Befehl PRINT#1 gegeben wird.

Anschliessend muss das File wieder geschlossen werden !

4.4.3 Ausgabe auf den Bildschirm

A. Ausgabe an eine bestimmte Stelle des Bildschirms

Am einfachsten kann eine Ausgabe auf den Bildschirm erreicht werden, indem der Bildschirmspeicher ($8000-83FF, bzw. $8000-87FF) direkt mit Zeichen im Bildschirmcode beschrieben wird.

Dieses Vorgehen entspricht dem BASIC-Befehl POKE und hat den Nachteil, dass ASCII-Zeichen zunächst in den Bildschirmcode umgewandelt werden müssen und dann erst auf den Bildschirm ausgegeben werden können.
Bei dieser Art der Ausgabe werden auch die Funktionen etwaiger Steuerzeichen nicht berücksichtigt.

Beim Bildschirm-RAM ist folgendes zu bemerken:
Der Bereich, der auf den Bildschirm geschrieben wird, befindet sich an vier verschiedenen Adressen im RAM:

a. von $8000 bis $83FF = 32768 bis 32791
b. von $8400 bis $87FF = 32792 bis 34815
c. von $8800 bis $8BFF = 34816 bis 35839
d. von $8C00 bis $8FFF = 35840 bis 36864

Dies ist auf die hardwaremässige Dekodierung des Bildschirm-RAMs zurückzuführen.
Das bedeutet, dass ein Beschreiben der Speicherzelle $8000 die gleiche Wirkung auf dem Bildschirm hervorruft wie das Beschreiben der Zellen $8400, $8800 oder $8C00.
(Beim 80-Zeichen-Bildschirm sind a+b identisch mit c+d)

Um diese Form der Ausgabe zu erleichtern, ist für den CBM 3000 in Anhang 8 ein Bildschirmspeicherplan zu finden.

Die Umwandlung ASCII nach Bildschirmcode entfällt bei Benutzung der Routinen, wie sie vom CBM-Betriebssystem verwendet werden.
Ausserdem ist durch sie die Bildschirmsteuerung durch Ausgabe von Steuerzeichen (Bildschirm löschen, Cursorfunktionen) möglich.

B. Ausgabe eines Strings

a. Ohne Angabe der Stringlänge: 51740=$CA1C

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt worden sein (Normalzustand nach Einschalten) !

Die Anfangsadresse des Strings wird folgendermassen übergeben:

LB im Akkumulator
HB im Y-Register.

WICHTIG!: Der String muss mit dem Byte $00 enden!

Ablauf der Routine

Die Routine gibt den Inhalt des Strings (ASCII!) ab der aktuellen Cursorposition auf den Bildschirm aus (mit richtiger Behandlung der Steuerzeichen).

b. Mit Angabe der Stringlänge: 51747=$CA23

(Diese Routine ist in a. enthalten)

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt worden sein (Normalzustand nach Einschalten) !

Die Anfangsadresse des Strings muss in den Speicherzellen $1F (LB) und $20 (HB) abgelegt sein. Das XR enthält die Anzahl der Zeichen, die ausgedruckt werden sollen.

Ablauf der Routine

Die Subroutine gibt die Zeichen des Strings (ASCII!!) auf den Bildschirm mit richtiger Behandlung der darin enthaltenen Steuerzeichen aus.

C. Akkuinhalt an Bildschirm	BSOUT	62002 < 62054

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt sein (Normalzustand nach Einschalten).

Der AC muss das auszugebende Zeichen enthalten.

Ablauf der Routine

Hierbei werden zunächst die Register (AC, YR, XR) auf den Stack gerettet.

Sodann erfolgt eine Abfrage, ob es sich beim Zeichen im Akkumulator um ein geshiftetes Zeichen (Bit 7 =1) oder um ein Steuerzeichen handelt.

Je nachdem werden dann Routinen angesprungen, die auf dem Bildschirm die entsprechende Steuerfunktion bewirken, bzw. das Zeichen ausgeben.

D. Ausgabe von 2 Zeichen	WRTWO	59268 < 55089

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt worden sein (Normalzustand nach Einschalten) !

Das erste (ASCII-) Zeichen steht im X-Register,
das zweite (ASCII-) Zeichen im Akkumulator.

Ablauf der Routine

Die Routine gibt die beiden Zeichen ab der aktuellen Cursorposition auf den Bildschirm aus.

E. Ausgabe von 4 Hexa-Zeichen	WROA	59242 < 55063

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt worden sein (Normalzustand nach Einschalten) !

Diese Routine ermöglicht die Ausgabe einer 2-Byte-Zahl oder - Adresse als vierstellige Hexadezimalzahl auf den Bildschirm.
Dazu muss die Adresse oder Zahl folgendermassen abgelegt sein:

LB nach $FB,
HB nach $FC.

F. Ausgabe eines Zeichens im Akkumulator

Voraussetzungen

Der IEC-Bus muss auf Bildschirm als Ausgabe-Gerät eingestellt worden sein (Normalzustand nach Einschalten) !

a. als ASCII-Zeichen: Routine OUTCH
65490 (BASIC 3+4)

b. als 2-stellige Hexadezimalzahl: Routine WROB
59253 < 55074

G. Ausgabe einer reellen Zahl — 56547 < 53133

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Bereich des FAC#1

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETRNM ($F272<F2A6).

Ablauf der Routine

Der Inhalt des FAC#1 wird als Fliesskommazahl auf den Bildschirm ausgegeben. Dazu muss der FAC#1 zuvor mit der auszugebenden Zahl im Registerformat geladen werden.

H. Bildschirmmanipulationen

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETRNM ($F272<F2A6).

a. Bildschirm nach oben schieben — 58687 < 58344

Der Bildschirminhalt wird um eine logische Zeile (40 oder 80 Zeichen) nach oben geschoben. Der Cursor wird auf den Anfang der letzten Zeile gesetzt.

b. Bildschirm löschen — 57897 < 57365

Der gesamte Bildschirminhalt wird gelöscht und der Cursor auf den Beginn der ersten Zeile (=Ecke oben links) gesetzt.

Zusätzliche Funktionen im BASIC 4.0

Die für BASIC 3.0 beschriebenen Bildschirmroutinen sind auch im BASIC 4.0 enthalten. Darüberhinaus ermöglicht jedoch der sogenannte 'SCREEN-Editor' einige weitere komfortable Möglichkeiten zur Bildschirmmanipulation.
Dieser 'SCREEN-Editor' ist ein ca. 2KB umfassendes Betriebssysstem für die Bildschirmsteuerung.
Sämtliche Steuerungen können auch von BASIC durch Ausgabe eines entsprechenden CHR$(X) durchgeführt werden.
Verwendbar sind davon die folgenden Routinen:

c. Bildfenster nach oben rollen — 58344=$E3E8

Der gesamte Inhalt des definierten Bildschirmfensters wird nach oben geschoben. Der Inhalt der obersten Zeile geht dabei verloren.
Entspricht dem BASIC- Befehl 'print chr$(25)'.

d. Bildfenster nach unten rollen — 58312=$E3C8

Der gesamte Inhalt des definierten Bildschirmfensters wird nach unten geschoben. Der Inhalt der obersten Zeile geht dabei verloren.
Entspricht dem BASIC- Befehl 'print chr$(153).

d. Zeilenende löschen — 57793=$E1C1

Der Inhalt der Zeile, in der sich der CURSOR befindet wird ab der Stelle (Y+1) gelöscht.

e. Zeilenanfang löschen — 58262=$E7A0

Der Inhalt der Zeile, in der sich der CURSOR befindet wird bis zur aktuellen Cursorposition (durch die Zeropageadresse 226 definiert) gelöscht.
Die Position des ersten bleibenden Zeichens muß daher in 226 angegeben werden.
Entspricht dem BASIC-Befehl 'print chr$(150)'.

f. Zeile entfernen 58087=$E2F3

Die aktuelle Cursor-Zeile, die in der Zeropageadresse 216 definiert ist, wird gelöscht und der Bildschirminhalt darunter nach oben nachgerückt. Entspricht dem BASIC-Befehl 'print chr$(21)'.

g. Zeile einfügen 58816=$E5C0

Die aktuelle Cursor-Zeile, die in der Zeropageadresse 216 definiert ist, wird freigemmacht und der Bildschirminhalt darunter nach unten gerückt. Entspricht dem BASIC-Befehl 'print chr$(149)'.

h. Bildfenster auf vollen Bildschirm setzen 57810=$E1D2

Die Pointer für das definierten Bildschirmfensters werden auf den vollen Bildschirm gesetzt.

i. Auf Textmodus umschalten 57466=$E07A

Der Bildschirm wird umgeschaltet in den Textmodus. Dies entspricht dem BASIC-Befehl 'print chr$(142)'.

k. Auf Grafikmodus umschalten 57464=$E078

Der Bildschirm wird umgeschaltet in den Grafikmodus. Dies entspricht dem BASIC-Befehl 'print chr$(14)'.

l. Glocke anschlagen 59047=$E6C1

Der eingebaute Lautsprecher wird angesprochen. Diese Routine wird vom Betriebssystem immer dann verwendet, wenn bei der Zeicheneingabe die 5. Stelle vor dem rechten Bildschirmrand erreicht wird.

I. Cursormanipulation (nur CBM 3000) 57949=$E25D

Wichtige Speicherzellen

Adresse	Bedeutung	
198=$C6	Cursorspalte	(0-39)
216=$D8	Cursorzeile	(0-24)

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch die Routine SETNRM ($F272)

Ablauf der Routine

Diese Routine berechnet die Stelle des Cursors im Bildschirm aus den Angaben für die Cursorspalte und die Cursorzeile.

Beispiel: Der Cursor soll auf Spalte 15 in Zeile 10 gesetzt werden:

```
                0001 ZEILE      .DE $D8       ;CURSORZEILE
                0002 SPALTE     .DE $C6       ;CURSORSPALTE
                0003 POSITION   .DE $E25D     ;CURSOR POSITIONIEREN
                0004 AUSDRUCK   .DE $F232     ;GIBT ZEICHEN IM AC
                0005            ;              HINTER DER AKTUELLEN
                0006            ;              CURSOR-POSITION AUS.
                0007            ;-----------------------------------
                0008            .BA $033A
033A- A9 0A     0009 ANFANG     LDA #10       ;ZEILENNUMMER
033C- 85 D8     0010            STA *ZEILE    ;SETZEN
033E- A9 0F     0011            LDA #15       ;SPALTENNUMMER
0340- 85 C6     0012            STA *SPALTE   ;SETZEN
0342- 20 5D E2  0013            JSR POSITION ;POSITION BERECHNEN
                0014            ;UND CURSOR POSITIONIEREN
0345- A9 43     0015            LDA #'C       ;ZEICHEN 'C'
0347- 20 32 F2  0016            JSR AUSDRUCK ;AUSGEBEN
034A- 60        0017            RTS
                0018            .EN
```

```
LABEL FILE:  ( / = EXTERNAL )

/ZEILE=00D8             /SPALTE=00C6            /POSITION=E25D
/AUSDRUCK=F232          ANFANG=033A
```

Bei Ausführung des Programmes wird in Zeile 10 auf die 15. Spalte das Zeichen 'C' auf den Bildschirm ausgegeben und dann mit dem aufrufenden Programm weitergefahren.

4.4.4 Meldungen des Betriebssytems

Die Meldungen des Betriebssystems sind als Zeichenketten ab den Adressen 49554 (>45581) bzw. ab 61440=$F000 (System 3+4) im ROM abgelegt und werden folgendermassen ausgegeben:

Das XR (bzw.YR) wird mit der Offsetadresse der Meldung geladen und sodann JMP 50007<46031 (bzw.JSR 61782<61829) ausgeführt. Die Routine des Betriebssystems holt sich nun XR-indiziert (YR-indiziert) ein Zeichen der Meldung nach dem anderen in den AC und fragt dabei ab, ob das Bit 7 dieses Zeichens gesetzt ist.

Ist es nicht gesetzt, so wird das Zeichen im AC ausgedruckt und nach Inkrementierung von XR (bzw.YR) das nächste Zeichen geholt.

Wenn das Bit 7 jedoch gesetzt ist, so löscht die Routine dieses Bit und druckt dann das Zeichen als letztes aus, um danach in den BASIC-Warmstart (50059 < 46079) zu springen.

Von den beiden beschriebenen Routinen ist nur die mit der Einsprungadresse 61782 <61829 als Subroutine ausgelegt:

> Die Meldungen, die im ROM im Bereich 61440-61621 =$F000-F0B5 abgelegt sind, können also durch Laden des YR mit der Offsetadresse der gewünschten Fehlermeldung ausgegeben werden.

Nach Ausdruck der Fehlermeldung erfolgt Rücksprung in das aufrufende Programm.

Die Anfangsadressen der Strings zur Ausgabe von Meldungen zeigt folgende Tabelle:

Adress-Offset	Meldung:
XR=$6B	BAD SUBSCRIPT
XR=$DB	CAN'T CONTINUE
YR=$74	DEVICE NOT PRESENT
XR=$85	DIVISION BY ZERO
XR=$F9	ERROR
XR=$BF	FILE DATA
YR=$24	FILE NOT FOUND
YR=$17	FILE NOT OPEN
YR=$0E	FILE OPEN
YR=$3D	FOR
XR=$C8	FORMULA TOO COMPLEX
YR=$A3	FOUND
XR=$95	ILLEGAL DIRECT
XR=$35	ILLEGAL QUANTITY
YR=$5F	LOAD
XR=$00	NEXT WITHOUT FOR
YR=$86	NOT INPUT FILE
YR=$94	NOT OUTPUT FILE
YR=$AA	OK
YR=$56	ON TAPE #
XR=$4D	OUT OF MEMORY
XR=$45	OVERFLOW
YR=$41	PRESS PLAY
YR=$4D	& RECORD
XR=$78	REDIM'D ARRAY
XR=$16	RETURN WITHOUT GOSUB
YR=$32	SEARCHING
XR=$B0	STRING TOO LONG
XR=$10	SYNTAX
YR=$00	TOO MANY FILES
XR=$A3	TYPE MISMATCH
XR=$E9	UNDEF'D FUNCTION
XR=$5A	UNDEF'D STATEMENT
YR=$6D	VERIFY
YR=$64	WRITING

4.4.5 Eingabe über den IEC-Bus

Für die Eingabe eines Zeichens vom IEC-Bus ist nur eine einzige Routine vorgesehen:

A. Eingabe vom IEC-Bus	IECINP	61836 < 61888

Wichtige Speicherzellen

Adresse	Bedeutung
210=$D2	Logische Filenummer
211=$D3	Sekundäradresse
212=$D4	Gerätenummer (Primäradresse)

Voraussetzungen

Das File, das durch die Routine angesprochen werden soll, muss vorher geöffnet und das adressierte Gerät als TALKER benannt sein.

Ablauf der Routine

Die Routine übernimmt ein Zeichen vom IEC-Bus in den AC und überwacht dabei den Ablauf des Transfers:

Zunächst wird das Status-Byte gelöscht (150:=0).
Danach wird abgefragt, von welchem Gerät die Routine ein Zeichen holen soll (Inhalt von 175=?).
Je nach Device-Nr. spricht die Routine die folgenden Geräte an:

0	= Tastatur
1/2	= Kassettenrekorder 1/2
3	= Bildschirm
>3	= IEC-Bus-Gerät

Es folgt eine auf das jeweilige Gerät zugeschnittene Behandlung der Eingabe:

0: Tastatur: Das erste Zeichen aus dem Tastaturpuffer (=623) wird geholt und in den AC übergeben. Die anderen Stellen des Tastaturpuffers (624-632) werden nach unten geschoben und der Pufferzähler (158) dekrementiert.

1/2: Kassettenrekorder: 1 Byte wird aus dem Kassettenpuffer des jeweiligen Rekorders als ein Zeichen gelesen.
Falls dieses Zeichen ein END OF FILE ist, wird das Statusbyte (in 150) geändert.
Das geholte Byte steht nun im AC zur Verfügung.

3: Bildschirm: Das nächste Zeichen hinter der aktuellen Cursorposition wird im AC bereitgestellt.

>3: IEC-Bus-Gerät: Ein Byte wird über den IEC-Bus vom Gerät mit der Device-Nummer in (175) geholt und im AC bereitgestellt.

Falls kein Zeichen von dem angesprochenen Eingabegerät gesendet wurde, so hat der AC nach Abschluss der Routine den Wert 0.

Bei einem Fehler wird das Statuswort in 150=$96 geändert und kann danach abgefragt werden.

Am besten ist die Wirkung der Routine aus dem Beispiel auf der nächsten Seite ersichtlich.

```
                    0001 CHAR        .DE $0360 ;ZWISCHENSPEICHER
                    0002             ;          FUER GEHOLTES ZEICHEN
                    0003 STATUS      .DE $96
                    0004 TALKER      .DE $F0B6  (<F0D2)
                    0005 UNTALK      .DE $F17F  (<F1AE)
                    0006 SEKADR      .DE $D3
                    0007 SENDADR     .DE $F128  (<F143) ;SENDET SEK.ADRESSE
                    0008 EINGABE     .DE $F18C  (<F1C0)
                    0009             ;-----------------------------
                    0010             .BA $033A
033A- A9 00         0011 ANFANG      LDA #0        ;STATUSWORT
033C- 85 96         0012             STA *STATUS   ; LOESCHEN
033E- 20 B6 F0      0013             JSR TALKER    ;GERAET = TALKER
0341- A5 D3         0014             LDA *SEKADR   ;SEKUNDAERADRESSE
0343- 20 28 F1      0015             JSR SENDADR   ;AN IEC-BUS SENDEN
0346- 20 8C F1      0016             JSR EINGABE   ;ZEICHEN HOLEN
0349- 8D 60 03      0017             STA CHAR      ;UND ABSPEICHERN
034C- 20 7F F1      0018             JSR UNTALK    ;GERAET=UNTALK
034F- A5 96         0019             LDA *STATUS   ;STATUSWORT
0351- D0 01         0020             BNE FEHLER    ;UEBERPRUEFEN
0353- 00            0021             BRK
                    0022 FEHLER      ;FEHLER BEI DER UEBERTRAGUNG
                    0023             ;BEHANDELN
                    0024             .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/CHAR=0360                /STATUS=0096              /TALKER=F0B6 (<F0D2)
/UNTALK=F17F (<F1AE)      /SEKADR=00D3              /SENDADR=F128 (<F143)
/EINGABE=F18C (<F1C0)     ANFANG=033A               FEHLER=0354
```

Wird dieses Programm aufgerufen, so wird ein Zeichen vom eingestellten Gerät (z.B. Floppy-Disk) geholt und in CHAR (=$0360) abgelegt und steht dann dort zur Weiterverwendung bereit.

Zur Eingabe von mehreren Zeichen können die Zeilen 16 und 17 mehrmals ausgeführt werden.
Die Kontrollfunktionen müssen lediglich vor und nach dem Datentransfer ausgeführt werden.

4.4.6 Eingabe über Tastatur und Bildschirm

4.4.6.1 Der Tastaturpuffer

Die Tastatur wird von der Interrupt-Routine abgefragt.
Wird eine Taste gedrückt, so wird das entsprechende Zeichen in die nächste freie Stelle des Tastaturpuffers (623-632) abgelegt und der Pufferzähler (158) dabei um 1 erhöht.
Beim GET- und INPUT-Befehl werden die Zeichen ebenfalls aus dem Tastaturpuffer geholt - und nicht direkt von der Tastatur selbst, wie man vielleicht annehmen sollte !

Der restliche Inhalt des Tastaturpuffers (soweit vorhanden) rückt dabei um eine Stelle nach vorne. Bei Beendigung eines Programmes und Rücksprung in den READY-Modus wird der Inhalt des Puffers auf den Bildschirm entleert.

Diese Eigenschaft des Tastaturpuffers wird verwendet, um den sogenannten DYNAMIC KEYBOARD zu realisieren, von dem viele Programme Gebrauch machen, um beispielsweise während dem Programmablauf DATA-Statements zu verändern oder Eingaben für die FN-Funktion zu ermöglichen:

> Hierfür wird der Tastaturpuffer (in Maschinensprache oder durch POKE-Befehle) mit den Zeichen des entsprechenden Befehles beschrieben, gefolgt vom Zeichen CARRIAGE RETURN (13=$0D).
> Nach Beenden des Programmes erscheint der Inhalt des Puffers auf dem Bildschirm und wird (wegen $0D) als BASIC-Befehl ausgeführt.
> Im Normalfall werden diese Möglichkeiten allerdings nicht sehr häufig genutzt.

Der Tastaturpuffer ermöglicht zusätzlich die Eingabe von Befehlen oder Daten schon einige Zeit BEVOR sie eigentlich vom Rechner verarbeitet werden können.

Dabei ist jedoch zu beachten, dass der Tastaturpuffer beim BASIC 3 nur bis zu 9 Zeichen aufnehmen kann. Wird das 10. Zeichen eingegeben, so wird der Pufferzähler auf 0 gesetzt und der Inhalt des Puffers damit gelöscht. Der Puffer wird dann von neuem gefüllt, währenddem die vorher eingegebenen Zeichen verlorengehen.

Beim BASIC 4 wird dagegen beim Erreichen der Puffergrenze der Puffer nicht gelöscht, sondern keine weiteren Zeichen angenommen. Ausserdem kann die Länge des Puffers (in $E3=227) bis zu 256 Zeichen betragen.

Aber Vorsicht !!
Der Puffer für Tape 1 wird dabei überschrieben, bei einer Pufferlänge von mehr als 201 Zeichen auch der Puffer für Tape 2. Dieser Bereich wird jedoch von den Disk- Basic-Befehlen verwendet !!!!

4.4.6.2 Eingabe-Routinen

A. Zeichen aus Tastaturpuffer holen GETASC 61905=$F1D1

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272 < F2A6).

Ablauf der Routine

Die Routine übergibt das Zeichen aus der untersten Stelle (623=$026F) des Tastaturpuffers (623-632 = $026F-0278) in den AC und erniedrigt den Zähler für den Inhalt des Puffers (158=$9E) um den Wert 1.
Wenn kein Wert im Tastaturpuffer enthalten ist, so enthält der AC den Wert 0.

Diese Art der Eingabe entspricht dem BASIC-Befehl GET.

Das Beispielprogramm wartet, bis die RETURN-Taste gedrückt wird:

```
                 0001             ;HOLT ZEICHEN VON TASTATUR
                 0002             ;------------------------------
                 0003             ;
                 0004             .BA $033A
033A- 20 D1 F1   0005 GET         JSR GETASC
033D- C9 0D      0006             CMP #$0D      ;=RETURN ?
033F- D0 F9      0007             BNE GET       ;NEIN:=> NOCHEINMAL
0341- 00         0008 WEITER      BRK
                 0009             ;PROGRAMMFORTSETZUNG, WENN
                 0010             CARRIAGE RETURN GEDRUECKT.
                 0011             .EN
```

```
LABEL FILE:  ( / = EXTERNAL )

/GETASC=F1D1 (<F205)      GET=033A                WEITER=0341
```

Der Einsprung in das Programm erfolgt mit SYS 826.
Erst wenn CARRIAGE RETURN eingegeben wird, fährt das Programm weiter.

In diesem Fall: Sprung in die Monitor-Warteschleife.

B. Eingabe einer Zeichenfolge — CHRSIN 61921 < 61973

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272 < F2A6).

Ablauf der Routine

Diese Routine erwartet die Eingabe eines Strings, der mit CARRIAGE RETURN ($0D) endet.
Wird die Routine wiederholt aufgerufen, so legt sie jeweils das nächste Zeichen in den AC ab. Das letzte Zeichen ist das Zeichen $0D.

Im angegebenen Beispiel ist die Adresse, ab der der einzugebende String abgelegt werden soll, vorher in ($1F,20) anzugeben. Für Tests bietet sich die Adresse $8000 an, da der String dann in die obere linke Bildschirmecke abgelegt wird.
Nach Einsprung mit SYS 826 erwartet das Programm die Eingabe von beliebigen Zeichen bis zum Auftreten eines CARRIAGE RETURN und legt diese ab der Anfangsadresse ab.
Der String kann dann mit Hilfe des Monitors ab der Anfangsadresse eingesehen werden.

```
                0001 CHRSIN     .DE $F1E1 (<F215) ;HOLT ZEICHEN VON
                                                  ;DER TASTATUR
                0002 POINTER    .DE $001F     ;GIBT AN, WOHIN DER
                                              ;STRING GESPEICHERT
                                              ;WERDEN SOLL
                0003            ;-----------------------------
                0004            .BA $033A
033A- A0 00     0005 ANFANG     LDY #0       ;ZAEHLER FUER DIE ANZAHL
                0006            ;DER EINGEGEBENEN ZEICHEN
033C- 20 E1 F1  0007 EINGABE    JSR CHRSIN
033F- C9 0D     0008            CMP #$0D      ;= CARRIAGE RETURN ?
0341- F0 05     0009            BEQ ENDE      ;JA:=> EINGABE BEENDET
0343- 91 1F     0010            STA (POINTER),Y
0345- C8        0011            INY  ;ZEICHENZAEHLER ERHOEHEN
0346- D0 F4     0012            BNE EINGABE  ;MAX. 255 ZEICHEN
0348- 60        0013 ENDE       RTS
                0014            .EN
```

```
LABEL FILE:  ( / = EXTERNAL )

/CHRSIN=F1E1 (<F215)     /POINTER=001F          ANFANG=033A
EINGABE=033C             ENDE=0348
```

C. Eingabe Hex-Byte vom Bildschirm HBYTIN 59318 < 55139

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272 < F2A6).

Ablauf der Routine

Die Routine übernimmt eine zweistellige Hexadezimalzahl als ASCII-Zeichen vom Bildschirm und übergibt den Wert nach Umwandlung hexadezimal in den AC.
Diese Hexa-Zahl wird ab der aktuellen Cursorposition gelesen. Wenn es ein Space ($20) ist, wird das Carry-Flag gesetzt.

D. Eingabe Hex-Adresse vom Bildsch. HADRIN 59303 < 55124

Wichtige Speicherzellen

Adresse	Bedeutung
251=$FB	Speicherzellen, in die die
252=$FC	Hex-Adresse abgelegt wird

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272 < F2A6).

Ablauf der Routine

Diese Routine liest eine 4-stellige Hexadezimalzahl vom Bildschirm ab der aktuellen Cursorposition und legt sie als 2-Byte-Adresse (HB/LB) in (251/252=$FB/FC) ab.
Dabei liest sie zunächst das HB und speichert es nach (251) und dann LB , um es nach (252) zu speichern.

Findet sie statt einer Zahl Spaces, so wird nichts verändert.
Die entsprechende Stelle (LB, HB oder beide) der Adresse bleibt dann unverändert.

Insbesondere kann durch Eingabe von 2 Spaces, gefolgt von einer 2-stelligen Hex-Zahl, nur das LB der Adresse verändert werden (das gleiche gilt für das HB).
Bei Auffinden von 4 Spaces erfolgt demnach keine Änderung des Zeigers.

E. Eingabe von: Dezimaladresse,Zahl GETNUM 54982 < 54982

Wichtige Speicherzellen

Adresse		Bedeutung
17=$11	(LB:)	Adresse, an die die Adresse vom
18=$12	(HB:)	Bildschirm abgelegt werden soll.

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272<F2A6)

Ablauf der Routine

Die Routine holt eine Adresse (0-65535), gefolgt von einem Wert (0-255) vom Bildschirm. Zwischen Adresse und Wert muss ein Komma stehen. Dabei überprüft sie, ob die Klammern (falls vorhanden) richtig gesetzt sind.
Die Adresse legt sie nach ($11/12) ab, der Wert wird im AC übergeben.

F. Addiert Zahl vom Bildsch. zum FAC#1: TIMNUM 51496 < 47531

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Bereich des FAC#1

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272<F2A6)

Ablauf der Routine

Die Zahl ab der aktuellen Cursorposition wird vom Bildschirm gelesen und zum FAC#1 addiert.

G. Eingabe Fliesskommazahl vom Schirm FRMEV1 52389 < 48542

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Bereich, in den die Zahl vom Bildschirm abgelegt wird (=FAC#1).

Voraussetzungen

Der IEC-Bus muss auf Ein/Ausgabe von Tastatur/Bildschirm eingestellt sein (=Normalzustand nach dem Einschalten, erreichbar durch Routine SETNRM ($F272<F2A6)

Ablauf der Routine

Diese Routine übernimmt eine Fliesskommazahl (oder auch Integerzahl) vom Bildschirm und legt sie im Registerformat im FAC#1 ab.

4.4.7 Ein/Ausgabe mit Kassettenrekorder

4.4.7.1 Ausgabeformat und Speicherbelegung

Der Kassettenrekorder nimmt eine Sonderstellung in der Gruppe der CBM-Peripherie-Geräte ein:

Im Gegensatz zu den anderen Geräten erfolgt der Datentransfer nicht 8-Bit-parallel, sondern seriell. Ausserdem sendet der Kassettenrekorder keine Steuerzeichen wie ein intelligentes Gerät (abgesehen von dem Signal TASTE GEDRUECKT).

Das hat zur Folge, dass der Rechner etwaige Uebertragungsfehler selbst erkennen muss. Ebenso muss gewährleistet sein, dass der Datentransfer auch genau in der Form erfolgt, wie sie der Kassettenrekorder erwartet.
Der CBM-Rechner übernimmt dabei sogar noch die Aufgabe eines Modems. Das heisst, er wandelt die binären Signale um in 2 verschiedene Frequenzen, bzw. empfangene Frequenzen in das decodierbare binäre Signal.

Die Daten werden im Kansas-City-Standard übertragen:
Dabei entspricht dem LOW-Pegel die Frequenz 1200 Hz und dem High-Pegel die Frequenz 2400 Hz.
Die Bitfolge wird mit einer Geschwindigkeit von 300 Baud (=300 Bit/Sekunde) gesendet bzw. empfangen.

Die Synchronisation beim Lesen vom Kassettenrekorder wird erreicht durch Verbiegen des IRQ-Vektors auf spezielle Bearbeitungsroutinen. Es muss deshalb der IRQ-Vektor auf seinen Normalwert (=$E62E) gestellt werden, bevor Kassettenoperationen aufgerufen werden.

Solange ein Datentransfer erfolgt, wird jeder Tastendruck bis auf die STOP-Taste ignoriert (wegen der neuen Adresse des IRQ-Vektors). Die Uhr wird dabei dennoch weitergestellt.

Dabei ist folgende interessante Nebenwirkung zu bemerken:
Sobald der Kassettenrekorder Daten sendet, beeinflusst er (wahrscheinlich durch die Wirkung seiner Datenleitung als Interrupteingang des Rechners) die IRQ-Routine.
Das hat zur Folge, dass die Taste '>' im Rhythmus der am Rekordereingang des Rechners ankommenden Daten abgefragt wird - ohne dass vorher ein Befehl zur Steuerung des Rekorders gegeben wurde. Die Taste hat also dann quasi eine Repeat-Funktion.
Will man zum Beispiel kontrollieren, ob eine Kassette noch frei ist, so legt man sie in den Rekorder ein und betätigt die PLAY-Taste und die Taste '>':
Wenn die Kassette schon bespielt ist, wird eine ganze Kette von ">"-Zeichen ausgegeben.

Für die Korrespondenz mit den Kassettenrekordern werden 2 Bereiche des Speichers benötigt, die sogenannten TAPE-PUFFER:

a. Für den Kassettenrekorder 1: 634-825 =$027A-0339

b. Für den Kassettenrekorder 2: 826-1017=$033A-03FB

In diesen Puffern sind die Daten enthalten, die auf den jeweiligen Rekorder ausgegeben oder von ihm gelesen werden sollen.

Die Datenblocks selbst enthalten wichtige Angaben, die vom Rechner zur weiteren Verarbeitung benötigt werden.
Nach Einlesen eines Blocks können diese Angaben im entsprechenden Kassettenpuffer betrachtet werden:

Das erste Byte eines jeden Datenblocks gibt an, welcher Typ von den folgenden 4 Dateiarten eingelesen wurde:

=1: Fileheader eines Programmes

=2: Datenblock

=4: Fileheader eines Datenblocks

=5: End Of Tape - Block

Die nächsten vier Bytes (2. bis 5. Byte) eines Fileheaders enthalten Startadresse (LB/HB) und Endadresse (LB/HB) des Kassettenpuffers (bei Daten) bzw. des Speicherbereiches (bei einem Programm).
Nach diesen 5 Bytes folgen die ASCII-Zeichen des Filenamens (max. 16).

Der eingelesene Filename kann mit folgendem Programm ausgedruckt werden:

```
LDA #00       ENDE DES FILENAMENS
STA $0290     (FUER REK.#1) MARKIEREN
LDA #$7F      LB DER ANFANGSADRESSE
LDY #$02      HB      DES FILENAMENS
JSR STROUT    FILENAMEN AUSGEBEN.
```

Um Daten ordnungsgemäss abspeichern zu können, benötigt der Rechner mehrere Informationen, die für die Rekorder an bestimmten Speicherstellen hinterlegt werden:

Bedeutung:	Adressen
Rekorder-Nummer	212=$D4
Adresse des Filenamens	218/219=$DA/DB
Länge des Filenamens	209=$D1
Startadresse	251/252=$FB/FC
Endadresse	201/202=$C9/CA
Prüfsumme	633=$0279
Flag für LOAD(=1) und VERIFY(=0)	157=$9D
Status-Byte	150=$96

Bei der Ausgabe von Daten werden nacheinander Blocks von je 192 Bytes Länge (=Inhalt des TAPE-PUFFERS) auf den Rekorder geschrieben.

Dabei wird jeder Block zweimal übertragen, getrennt von einem kurzen Synchronisiersignal.

Beim Lesen wird dann der erste Block eingelesen und der zweite (mit gleichem Inhalt) zur Kontrolle der Richtigkeit des Transfers mit dem ersten verglichen.
Der CBM kann durch dieses Verfahren bis zu 30 fehlerhafte Zeichen korrigieren !

Bei Programmen läuft der gleiche Vorgang ab, mit der Ausnahme, dass dem Fileheader-Block zwei identische Programmblöcke folgen, die NICHT in 192 Byte-Abschnitte aufgespalten, sondern als Ganzes übertragen werden. Auch die Programmblöcke werden also zur Fehlererkennung und -korrektur zweimal hintereinander geschrieben.

Wie die Folge der Blocks auf dem Band aussieht (Synchronisierungssignal, Fileheader, Programmblock, usw.), ist im COMMODORE-Handbuch ausführlich beschrieben.

Beim Einlesen eines Programmes wird nur der Fileheader in den TAPE-Puffer übernommen.
Das Programm hinter dem Fileheader wird direkt in den Bereich abgespeichert, der durch die Angaben der Anfangs- bzw. Endadresse definiert wird.

Die auf den nächsten Seiten beschriebenen Routinen ermöglichen das Einlesen und Schreiben von (bzw. auf) Band auch mit anderen Formaten bzw. Speicherbereichen als sie der Rechner verwendet. Allerdings können die so bearbeiteten Daten auch nur mit speziellen (eigenen) Routinen behandelt werden.

4.4.7.2 Kassettenrekorder-Operationen

A. Nächsten Fileheader suchen	SUFIHD	62886 < 62949

Wichtige Speicherzellen

Adresse	Bedeutung
212=$D4	Rekorder-Nummer
214/215=$D6/D7	Startadresse des TAPE-Puffers

Ablauf der Routine

Diese Routine sucht auf dem gewählten Kassettenrekorder ($D4) nach dem nächsten Programmkopf (=Fileheader) und liest diesen in den Puffer ein. Danach gibt sie den Text "Found " und den Namen des gefundenen Files aus.

B. Block mit 192 Bytes Länge einlesen	BLCKRD	63573 < 63642

Wichtige Speicherzellen

Adresse	Bedeutung
212=$D4	Rekorder-Nummer (1 oder 2)
214/215=$D6/D7	Startadresse des TAPE-Puffers (Rek.#1:$027A, Rek.#2:$033A)

Ablauf der Routine

Diese Routine liest einen Block vom Band in den jeweiligen Kassetten-Puffer ein. Insbesondere können hiermit Programm- und Datenheader sowie Datenfiles eingelesen werden.

C. Einlesen eines Blocks beliebiger Länge in den Speicher

Wichtige Speicherzellen

Adresse	Bedeutung
212=$D4	Rekorder-Nr.
251/252=$FB/FC	Anfangsadresse des Speicherbereiches.
201/202=$C9/CA	Endadresse +1 des Speicherbereiches.

Ablauf der Routine

Um einen Block beliebiger Länge einzulesen wird folgendermassen vorgegangen:

1. Aufruf von Routine BLCKR1 63582 < 63651

   Diese Routine liest einen Block beliebiger Länge in den Speicherbereich zwischen Anfangs- und Endadresse ein.
   Um den Lesevorgang zu beenden, erfolgt ein

2. Aufruf von Routine ENDTAP 63733 < 63802

   Diese Routine beendet die Eingabe vom Kassettenrekorder, indem sie den Motor abstellt, den IRQ-Vektor wiederherstellt und die Aus/Eingabe auf Bildschirm/Tastatur einstellt. Die Tastatur wird zum aktiven Eingabegerät: ($0E)=0.
   Danach erfolgt Sprung in die BASIC-Routine STOP.

Die Routinen erlauben es, ein Programm in einen beliebigen Speicherbereich zu laden:

Nach Bestimmung des Rekorders (z.B. ($D4)=1) muss zunächst der Fileheader des Programmes geladen werden (=JSR SUFIHD). Im Puffer des Rekorders #1 stehen nun die Angaben über das dem Fileheader folgende Programm (siehe 4.4.7.1).
Über ein Maschinenprogramm (!) können die Zeiger für die Anfangs- und Endadresse des Programmes ($FB,FC und C9,CA) auf einen beliebigen Speicherbereich gesetzt werden.
Dorthin wird dann bei Aufruf der Routine BLCKR1 das Programm eingelesen und mit JSR ENDTAP die Eingabe abgeschlossen.

D. Laden eines Programmes	LODPRG	62409 < 62472

Wichtige Speicherzellen

Adresse	Bedeutung
241=$D4	Rekorder-Nr.
218/219=$DA/DB	Adresse des Filenamens
209=$D1	Länge des Filenamens
40/41 =$28/29	Programmanfang
42/43 =$2A/2B	Programmende

Voraussetzungen

Die Speicherzellen für die Rekorder-Nr., die Adresse und Länge des Filenamens müssen vor Aufruf dieser Routine gesetzt werden.

Ablauf der Routine

Diese Routine lädt ein Programmfile in den Speicher und setzt dabei die Zeiger auf den Programmanfang beziehungsweise den Anfang der Variablentabelle (=Programmende).
Die Angaben für den Bereich, in den das Programm abgelegt werden soll, sind im Fileheader enthalten (siehe 4.4.7.1).
Nach dieser Routine kann mit dem BASIC-Befehl RUN das Programm gestartet werden, bzw. bei Maschinenprogramm mittels Aufruf entsprechender Befehle oder Routinen.

E. Abspeichern eines Programmes	SAVPRG	63140 < 63203

Wichtige Speicherzellen

Adresse	Bedeutung
212=$D4	Rekorder-Nr.
218/219=$DA/DB	Adresse des Filenamens
209=$D1	Länge des Filenamens
251/252=$FB/FC	Startadresse
201/202=$C9/CA	Endadresse

Voraussetzungen

Die oben angegebenen Speicherzellen müssen vor Aufruf der Routine gesetzt werden!

Ablauf der Routine

Diese Routine schreibt den Speicherbereich zwischen Start- und Endadresse als Programm mit dem angegebenen Filenamen auf den eingestellten Kassettenrekorder.
Dabei werden keine Zeiger verändert, d.h. das aufrufende Programm kann bedenkenlos fortgesetzt werden.

F. Speicherbereich auf Band schreiben WRTRAN 63631 < 63701

Wichtige Speicherzellen

Adresse	Bedeutung
212=$D4	Rekorder-Nr.
251/252=$FB/FC	Startadresse
201/202=$C9/CA	Endadresse
632=$0279	Prüfsumme

Voraussetzungen

Die Rekorder-Nr. sowie Start- und Endadresse müssen vor Aufruf dieser Routine gesetzt werden !

Ablauf der Routine

Diese Routine schreibt einen Speicherblock zwischen Start- und Endadresse auf Band. Zeiger werden dabei nicht verändert.
Es empfiehlt sich vor Verwendung dieser Routine das Unterprogramm TAPEND aufzurufen (stellt IRQ-Vektor her, beendet etwaige Ein/Ausgabe). Die Prüfsumme muss auf den Wert 105=$69 gesetzt werden !

G. Ein/Ausgabe beenden TAPEND 63718 < 63787

Ablauf der Routine

Diese Routine wartet das Ende der Ein/Ausgabe ab und überprüft dabei die STOP-Taste. Nach Beendigung des Datentransfers wird der IRQ-Vektor wieder auf seinen normalen Wert ($E62E < E455) gesetzt.

H. Tasten des Rekorders überprüfen

Wichtige Speicherzelle

Adresse	Bedeutung
212=$D4	Rekorder-Nr.

Um zu überprüfen, ob eine Taste des Kassettenrekorders gedrückt ist, gibt es folgende Routinen:

a. vor einem Lesevorgang: 63506 < 63575

Ablauf der Routine

Die Routine überprüft, ob eine Taste des Rekorders gedrückt ist.
Wenn nicht, so gibt sie "PRESS PLAY ON TAPE #" (Rekorder-Nr) aus und quittiert anschliessendes Betätigen mit "OK".
Ansonsten wartet sie, bis eine Rekorder-Taste oder die STOP-Taste gedrückt werden.

b. vor einem Schreibvorgang: 63559 < 63600

Ablauf der Routine

Die Routine überprüft, ob eine Taste des Rekorders gedrückt ist.
Wenn nicht, so gibt sie "PRESS PLAY & RECORD ON TAPE #" (Rekorder-Nr.) aus und quittiert anschliessendes Betätigen der Taste mit "OK".
Ansonsten wartet sie, bis eine Rekorder-Taste oder die STOP-Taste gedrückt werden.

c. ohne Ausgabe eines Textes: 63541 < 63610

Ablauf der Routine

Die Routine prüft, ob eine Taste des eingestellten Kassettenrekorders gedrückt ist.
Bei der Rückkehr ist das ZERO-Flag gesetzt, falls der Test positiv ausgefallen ist.

4.4.7. Disk-Handling im BASIC 4.0

Der folgende Teil der Systembeschreibung gilt nur für Rechner der Serien CBM 8000 und 4000 mit BASIC 4. Diese enthalten - im Gegensatz zum CBM 3000 ein Disketten-Betriebssystem, das eine relativ einfache Bedienung der Floppy-Disks gestattet. Der CBM 3000 dagegen ist nur für den Betrieb mit Kassettenrekordern komfortabel ausgestattet. Das Floppy-Disk-Handling ist bei ihm etwas umständlich, aber auch für ihn werden ROMs angeboten, die diesen Komfort dennoch ermöglichen.

Wie aus dem Handbuch des CBM 3000 zu ersehen ist, werden Kommandos an die Floppy-Disk in einem String über den IEC-Bus gegeben.
Vor einer Uebergabe muss ein File eröffnet worden sein (siehe Kassettenrecorder-Handling). Die Fehlermeldungen der Floppy müssen über einen weiteren Kanal über INPUT gelesen werden.

Das DOS (=Disk Operating System) des COMMODORE-BASIC 4.0 baut sich nun die Kommandostrings für die Diskettenstation selbst auf und übergibt sie über den IEC-Bus an die Floppy-Disk.
Dabei werden auch das vorherige Oeffnen eines Files und Einlesen der Fehlermeldungen sowie das Schliessen des Files automatisch durchgeführt. Es werden die gleichen Kommandos und Strings verwendet, wie sie beim BASIC 3.0 der Anwender programmieren musste.

Beim Befehl 'COPY d0,"Name1" to d1,"Name2" werden beispielsweise die folgenden Befehle und Strings nacheinander vom DOS generiert und auf den IEC-Bus ausgegeben:

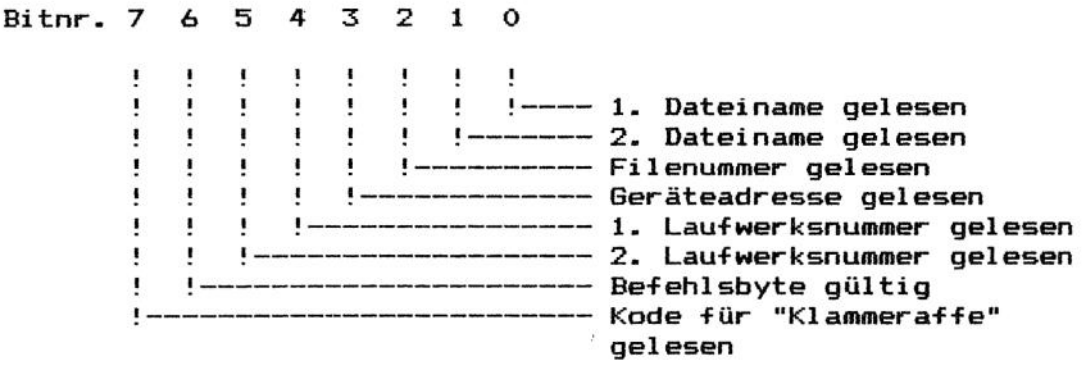
a) Einen Kanal für die Floppy öffnen.
b) Den String "c1:Name2=0:Name1" übergeben.
c) Fehlermeldung und Quittierung der Floppy lesen.
d) Im Direktmodus Fehler- und Quittierungsmeldung auf den Bildschirm ausgeben.
e) Kanal schliessen.

Die Diskettеninhaltsverzeichnisse können gelesen werden, ohne dass dabei ein etwa vorhandenes Programm im Arbeitsspeicher zerstört wird.
Auf die erweiterten Möglichkeiten des BASIC 4.0 gegenüber dem BASIC 3.0 soll hier nicht weiter eingegangen werden. Vielmehr sollen Teile aus diesem Betriebssystem für das eigene Diskettenhandling in Maschinensprache herausgegriffen und detailliert erklärt werden.

Die im Kommandostring geforderten Trennungszeichen und Parameter werden überprüft, wobei die Reihenfolge durch spezielle Codes für jeden Parameter innerhalb eines Disk-Befehles in einer Tabelle (55352 bis 55410) festgelegt ist.
Der ordnungsgemässe Aufbau und die dafür notwendigen Parametereingaben werden durch Kontrolle verschiedener SYNTAX-FLAGS im SYNTAX-Byte (830 = $033E) erreicht. Falls Parameter fehlen erfolgt die Meldung 'SYNTAX ERROR'.

Das SYNTAX-BYTE (830 = $033E)

Das SYNTAX-Byte enthält Informationen darüber, ob die erforderlichen Parameter für einen Disk-Befehl ordnungsgemäss eingegeben worden sind. Es enthält 8 Flags, deren Bedeutung folgendermassen kodiert ist:



Wenn eine Bedingung während des Aufbaus und der Parameterabfrage erfüllt wird, so wird gleichzeitig das zugehörige Bit gesetzt. Für jeden Disk-Befehl muss eine bestimmte Bitfolge vorhanden sein, deren Richtigkeit in der Syntax-Prüfung (55300-55351) erfolgt. Ist der Inhalt des SYNTAX-Bytes für den zu überprüfenden Disk-Befehl nicht korrekt, so erfolgt im Direktmodus 'SYNTAX ERROR', im Programm-Modus wird der Diskstatus DS und DS$ entsprechend gesetzt.

Zusammenbau des Befehlsstrings	56314 = $DBFA

In der Tabelle (55352-55410) steht für jeden der Diskbefehle eine Reihe von Codes.
Diese Codes haben für die Zusammensetzung des Befehlsstrings für die Floppy unterschiedliche Bedeutungen:

a) Codes kleiner als 128 werden direkt als ASCII-Zeichen in den String übernommen.

b) Codes grösser als 128 haben die Funktion eines Platzhalters für bestimmte Parameter, die für den String benötigt werden:

```
208   = Disk-ID
209   = erste Drive-Nummer
210   = zweite Drive-Nummer
224   = Dateityp
225   = Dateiart
241   = erster Dateiname
242   = zweiter Dateiname
255   = "Klammeraffe" Zeichen
```

In der Tabelle sind sozusagen symboliscche Befehlsstrings abgelegt.
Findet der Interpreter einen Code grösser als 128, so springt er in die entsprechende Subroutine
(z.B.: "baue Disk-ID in Befehlsstring ein")
und fährt danach fort.
Für den Befehl 'RENAME' beispielsweise sind in 55399-55406 die Codes zu finden:

```
 82   = 'r'
209   = erste Drivenummer
 58   = ':'
242   = erster Dateiname
 61   = '='
209   = erste Drivenummer
 58   = ':'
241   = erster Dateiname
```

Diese Codes steuern nun den Zusammenbau des Befehlsstrings, der ab Adresse 851 (=$0353) angelegt wird.
Die Adresse des Befehlsstrings (=851) befindet sich in (218,219) und seine Länge in (209).

Rückkehr aus der Routine erfolgt mit der Stringlänge in XR.

Einbau von Parametern in den Befehlsstring

Um Dateinamen oder die ID für den Befehlsstring zu definieren, werden diese zunächst im RAM abgelegt. Sie können dann mit den folgenden Routinen in den Befehlsstring eingebaut werden:

a) Dateinamen 1 einbauen — 56020 = $DAD4

Voraussetzung

Der Dateiname muss im RAM abgelegt sein. Seine Anfangsadresse befindet sich in (218,219) und seine Länge in (209).

Ablauf der Routine

Der definierte String wird als erster Dateiname in den Befehlsstring eingebaut.

b) Dateinamen 2 einbauen — 56033 = $DAE1

Voraussetzung

Der Dateiname muss im RAM abgelegt sein. Seine Anfangsadresse befindet sich in (253,254) und seine Länge in (209). Steht der Dateiname im RAM direkt hinter dem ersten Dateinamen und wurden die Pointer (253,254) bzw. (209) zwischenzeitlich nicht verändert, so brauchen nach Aufruf der Routine a. diese Pointer für die Routine b. nicht gesetzt werden, da sie vom Betriebssystem automatisch korrigiert werden.

Ablauf der Routine

Der definierte String wird als zweiter Dateiname in den Befehlsstring eingebaut.

c) Disk-ID einbauen — 56061 = $DCFD

Voraussetzung

Die Zeichen für die Disk-ID müssen folgendermassen im RAM abgelegt werden:
Das 1.Zeichen befindet sich in (831=$033f),
das 2.Zeichen in (832=$0340).

Ablauf der Routine

Die beiden definierten Zeichen werden als Disk-ID in den Befehlsstring eingebaut.

Ausführung des Disketten-Befehlsstrings	55437 = $D88D

Voraussetzungen

Der Ausgabekanal muss in (176) definiert sein (Bildschirm=3, Drucker=4).
Der String muß (z.B. von einer vorangegangenen Operation) definiert worden sein.

Ablauf der Routine

Der Disk-Befehlsstring ab $0353 (=851) wird ausgeführt. Nützlich zur Wiederholung einer vorangegangenen Disk-Operation !

RECORD-Zeiger einstellen	55437 = $DA31

Voraussetzungen

Es müssen die folgenden Paramter festgelegt wosrden sein: Filenummer in (210), Nummer des Records in (17/18), Stelle des gewünschten Zeichens im Record in (826).

Ablauf der Routine

Der Record-Zeiger wird auf das angegebene Zeichen eingestellt.

Diskstatus nach ds$ übernehmen	55687 = $D97D

Voraussetzung

Die Geräteadresse muss in (212) festgelegt wosrden sein:

Ablauf der Routine

Der Status des angegebenen IEC-Bus-Gerätes (üblicherweise Floppy) wird in die Statusvariable ds$ übernommen, die ERROR-Lampe der Floppy wird dabei gelöscht, falls sie eingeschaltet war.

Rückfrage	56227 = $DBA3

Voraussetzung

Das CARRY-Flag muss vor Einsprung in diese Routine gesetzt werden.

Ablauf der Routine

Die Routine gibt die Frage 'are you sure ?' aus und erwartet die Eingabe der Antwort 'yes', bzw. einen Teil davon (z.B.'y'). Wird diese Antwort eingegeben, so erfolgt Rückkehr mit CARRY-Flag = 0, ansonsten bleibt das CARRY-Flag gesetzt.

Meldung "BAD DISK"	56284 = $DBDC

Ablauf der Routine

Gibt die Meldung "BAD DISK" aus.

Vom verbleibenden Teil des Disk-Betriebssystemes liessen sich zwar einige weitere Subroutinen verwenden, jedoch müssten dafür so viele Parameter beachtet und definiert werden, dass es einfacher und übersichtlicher wird, wenn man die Routinen selbst aufbaut. Dabei kann dann ein intensives Studium der im Betriebssystem verwendeten Routinen sehr nützlich sein.

4.5 Der CBM-USER-Port

Neben dem sehr interessanten und universellen IEC-Bus besitzt der CBM noch einen sogenannten USER-Port, der etwas einfacher zu bedienen ist und ein nicht so kompliziertes Handshaking benötigt.

Der USER-Port ist nichts anderes als ein Teil der Daten- und Steuerleitungen des VIA 6522 (versatile interface adapter ,was soviel wie 'universeller Interface Anschluss' bedeutet).
Es sind hier der komplette Datenport A und die Datenleitung PB3 des Ports B (Cass.Write), die Steuerleitungen CA1, CB1 (Cass.2 Read) und CB2 erreichbar, sowie über Tricks auch der PA7 des PIA 6520 (im CBM als Tastatur-Ansteuerbaustein verwendet) am DIAGNOSTIC - Ausgang (siehe Beispiel A.).

Zur Ausgabe über den USER-Port sei folgendes angemerkt:

Das Betriebssystem verwendet die 6522-VIA (Port B) zur Ansteuerung der Kassettenrekorder-Motoren, zur Abfrage der PLAY-TASTE und zum Datentransfer zwischen den Rekordern und dem Rechner.
Der USER-Port selbst wird durch das Betriebssystem nicht bedient, so dass es auch keine verwendbaren Subroutinen im ROM gibt.

Wie man den USER-Port in BASIC ansprechen kann, wird in den Handbüchern zumindest einführend beschrieben.
Meistens wird vom Hobbyprogrammierer die Möglichkeit der Tonerzeugung am CB2 - Ausgang genutzt, die in einem kleinen Beispiel (B.) beschrieben werden soll.

Ausserdem ist wohl die am meisten genutzte Möglichkeit zum Datentransfer über den User-Port die folgende:

Durch Beschreiben des Datenrichtungsregisters DDRA mit einem bestimmten Bitmuster werden die Datenleitungen als Aus- oder Eingänge definiert:

Beschreiben mit %11100000 (=$E0) bedeutet z.B.:
Pins 1-5=Eingang, 6-8=Ausgang.
Der Zustand der Datenleitungen kann durch Lesen des Inhaltes des Ausgaberegisters für Port A abgefragt werden.

Um die Möglichkeiten des VIA 6522 voll ausschöpfen zu können, wird sich ein gründliches Studium der bei den Herstellern des Bausteins (Rockwell, MOS-Technology und Synertec) und ihren Distributoren erhältlichen Datenblättern nicht umgehen lassen. Eine eingehende Behandlung würde den Rahmen dieser Abhandlung sprengen, so dass nur auf die im CBM codierten Adressen der Register der VIA 6522 hingewiesen werden soll:

4.5.1 Die Registeradressen des VIA 6522

(Identisch für alle COMMODORE-Rechner)

```
59456=$E840   ORB  =Ausgaberegister Port B
59457=$E841   ORA  =Ausgaberegister Port A
                    (mit Handshake)
59458=$E842   DDRB =Datenrichtungsregister für Port B
59459=$E843   DDRA =Datenrichtungsregister für Port A

              T I M E R   1 :

59460=$E844   T1C-L=Lesen Zähler LOW,
                    Schreiben Vorspeicher LOW
59461=$E845   T1C-H=Lesen und Schreiben
                    Zähler HIGH
59462=$E846   T1L-L=Lesen und Schreiben in
                    Vorspeicher LOW
59463=$E847   T1L-H=Lesen und Schreiben in
                    Vorspeicher HIGH

              T I M E R   2 :

59464=$E848   T2C-L=Lesen Zähler LOW,
                    Schreiben Vorspeicher LOW
59465=$E849   T2C-H=Lesen und Schreiben

59466=$E84A   SR   =Schieberegister
59467=$E84B   ACR  =Hilfsregister
59468=$E84C   PCR  =Steuerregister
59469=$E84D   IFR  =Interrupt-Anzeigeregister
59470=$E84E   IER  =Interrupt-"Erlaubnis"-Register
59471=$E84F   ORA  =Ausgaberegister für Port A
                    o h n e    Einfluss auf
                    das Handshakesignal.
```

4.5.2 Beispiele zum User-Port

```
                    0001              ;BEISPIEL A.:VERWENDUNG DES
                                      ;             DIAGNOSTIC-SENSE
                    0002              ;ALS ZUSAETZLICHER EIN/AUSGANG:
                    0003              ;-----------------------------------
                    0004              ;
                    0005 CRA6520      .DE 59409     ;CONTROL REGISTER A
                    0006              ;              DER PIA 6520
                    0007 DDRA         .DE 59408     ;DATA DIRECTION
                    0008              ;              REGISTER A
                                      ;              DER PIA 6520
                    0009              ;-----------------------------------
                    0010              ;
                    0011              ;BEI JEDEM EINSPRUNG IN DAS PROGRAMM
                    0012              ;WIRD DER DIAGNOSTIC-SENSE ZWISCHEN
                    0013              ;EIN- UND AUSGANG UMGESCHALTET.
                    0014              ;
                    0015              ;ER KANN DANN Z.B. IN BASIC MIT
                    0016              ;?(PEEK(59408)AND128)/128 GELESEN
                    0017              ;ODER MIT POKE(59408),0 (BZW.128)
                    0018              ;GELOESCHT (BZW.GESETZT) WERDEN.
                    0019              ;-----------------------------------
                    0020              .BA $033A
033A- 78            0021 ANFANG       SEI  ;STOPPE TASTENABFRAGE
033B- A9 38         0022              LDA #%00111000  ;BIT2 DES CRA
033D- 8D 11 E8      0023              STA CRA6520     ;AUF 0 SETZEN
0340- AD 10 E8      0024              LDA DDRA         ;BIT7 DES DDRA
0343- 49 80         0025              EOR #%10000000   ;UMSCHALTEN
0345- 8D 10 E8      0026              STA DDRA
0348- A9 3C         0027              LDA #%00111100   ;BIT 2 DES CRA
034A- 8D 11 E8      0028              STA CRA6520      ;AUF 1 SETZEN
034D- 58            0029              CLI  ;TASTENABFRAGE FREIGEBEN
034E- 60            0030              RTS  ;ZURUECK INS AUFRUFENDE
                    0031              .EN  ;PROGRAMM
```

```
LABEL FILE:   ( / = EXTERNAL )

/CRA6520=E811          /DDRA=E810          ANFANG=033A
```

Manchmal wird ein zusätzlicher Ein/Ausgang am User-Port benötigt um ein bestimmtes Gerät überhaupt anschliessen zu können.
Dieses Maschinenprogramm ermöglicht die Nutzung des DIAGNOSTIC SENSE (Pin 5) am User-Port als Ein/Ausgang.

Das folgende Programm dient zur Tonerzeugung am CB2-Port (Toninterface erforderlich!).
Nach Einsprung mit SYS 826 beginnt die Tonausgabe, die mit der STOP-Taste abgebrochen werden kann.

```
                0001            ;BEISPIEL B.:TONERZEUGUNG
                                ;            AM PIN CB2
                0002            ;-----------------------------------
                0003            ;
                0004            ;REGISTER DES VIA 6522:
                0005            ;
                0006 ACR        .DE 59467     ;HILFSREGISTER
                0007 SR         .DE 59466     ;SCHIEBEREGISTER
                0008 T2CL       .DE 59464     ;TIMER 2: VORSPEICHER LOW
                0009            ;
                0010            ;-----------------------------------
                0011            .BA $033A
033A- A9 10     0012 ANFANG     LDA #$10      ;BETRIEBSZUSTAND DES VIA
                0013            ;              =FREI OSZILLIERENDES
033C- 8D 4B E8  0014            STA ACR       ; SCHIEBEREGISTER
033F- A9 0F     0015            LDA #%00001111        ;INHALT DES SR
                                                      ;FESTLEGEN:
0341- 8D 4A E8  0016            STA SR        ;BESTIMMT DIE TONHOEHE.
0344- A0 00     0017            LDY #$00
0346- 8C 48 E8  0018 TON        STY T2CL      ;BESTIMMT DIE FREQUENZ
0349- 98        0019            TYA
034A- 48        0020            PHA           ;YR RETTEN
034B- A0 01     0021            LDY #$01
034D- 20 5B 03  0022            JSR SCHLEIFE ;WARTEN
0350- 68        0023            PLA
0351- A8        0024            TAY           ;YR WIEDER HOLEN
0352- C8        0025            INY
0353- 20 01 F3  0026            JSR $F301 (<F335)  ;STOP-TASTE GEDRUECKT?
0356- D0 EE     0027            BNE TON       ;NEIN:WEITER
0358- 4C 64 03  0028            JMP ENDE
035B- A2 CB     0029 SCHLEIFE   LDX #$CB      ;ZEITSCHLEIFE
035D- CA        0030 MARKE      DEX           ;CA. 1 MS * (YR)
035E- D0 FD     0031            BNE MARKE
0360- 88        0032            DEY
0361- D0 F8     0033            BNE SCHLEIFE
0363- 60        0034            RTS
0364- A9 00     0035 ENDE       LDA #$00      ;REGISTER WIEDER
0366- 8D 4B E8  0036            STA ACR       ;ZURUECKSETZEN :
0369- 8D 4A E8  0037            STA SR
036C- 8D 48 E8  0038            STA T2CL
036F- 60        0039            RTS
                0040            .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/ACR=E84B              /SR=E84A              /T2CL=E848
ANFANG=033A            TON=0346              SCHLEIFE=035B
MARKE=035D             ENDE=0364
```

5. Mathematik

Das CBM-Betriebssystem besitzt eine Vielzahl mathematischer Funktionen, die es dem Benutzer erlauben, mathematische Problemstellungen komfortabel und schnell in BASIC zu programmieren.
Auch dem engagierten Assemblerprogrammierer wird durch diese implementierten Betriebssytemroutinen die zumeist mühselige und zeitraubende Arbeit abgenommen, eine mathematische Routine in Maschinensprache zu entwickeln.
Die Handhabung dieser Routinen soll im folgenden dem Anwender nahegebracht werden.

5.1 Zahlendarstellung

Der Mikroprozessor 6502 besitzt zwei Möglichkeiten der Zahlendarstellung, erstens durch die unmittelbare, binäre Wiedergabe, und zweitens durch die Wiedergabe im BCD-Kode ("binary-coded dezimal").
Das Prinzip zur Wiedergabe von Zahlen im BCD-Kode besagt, dass jede Stelle für sich zu kodieren ist und dass so viele Bits benutzt werden, wie zur vollständigen Zahlendarstellung notwendig sind. Um alle Ziffern von 0 bis 9 zu erfassen, benötigt man vier Bits. Dabei ist festzuhalten, dass sechs von den mit vier Bits möglichen 16 Kombinationen bei der BCD-Darstellung nicht benutzt werden. Der Vorteil von BCD-Zahlen liegt darin, dass sie vollkommen richtige Ergebnisse liefern können. Der Nachteil liegt darin, dass zu ihrer Darstellung sehr viel Speicherplatz benötigt wird und dass die Rechenoperationen relativ langsam ablaufen. Deshalb wird dieses Format auch nur in speziellen Fällen verwendet, in denen die Rechenzeit keine so grosse Rolle spielt und hoher Wert auf Genauigkeit gelegt wird.
In den bekannten Klein-Rechnern mit 6205-CPU wie z.B. APPLE, AIM65, PC100 u.s.w. wird dieser Dezimalmodus nicht verwendet.
Am häufigsten werden numerische Daten rein binär dargestellt, wie auch im CBM-Betriebssystem.
Das COMMODORE-BASIC hat zwei Arten der internen Zahlendarstellung:

a. Festkomma-Ganzzahlen (Integerzahlen) von -32768 bis +32767.
b. Fliesskomma-Zahlen von rd. 10^{-38} bis 10^{+38} mit maximal 9 signifikanten Stellen.

5.1.1 Ganze Zahlen

Die ganzen Zahlen (Integer-Zahlen) werden im 2er-Komplement in zwei Bytes dargestellt. Das höchstwertige Bit enthält das Vorzeichen.

Beispiele für ganze Zahlen

```
0001 =     +1
0100 =   +256
7FFF = +32767
FFFF =     -1
8000 = -32768
```

5.1.2 Fliesskommazahlen

Das Grundprinzip ist hier, dass die Dezimalzahlen mit einem festen Format wiedergegeben werden müssen.
Um hierbei keine Bits zu verschenken, werden in der Darstellung alle Zahlen normalisiert. So verschenkt z.B. 0,000123 auf der linken Seite der Zahl drei Nullen, die keine andere Bedeutung haben als die Stelle des Kommas anzugeben. Normalisieren dieser Zahl ergibt die Darstellung $0,123*10^{-3}$:"0,123" wird als normalisierte Mantisse, "-3" als Exponent bezeichnet. Die normalisierte Zahl gewinnt man bei diesem Beispiel durch das Streichen der bedeutungslosen Nullen links von ihr und entsprechendes Anpassen des Exponenten.
Allgemein erhält eine normalisierte Zahl die Form $M*10^{E}$, wobei M die Mantisse und E der Exponent ist.
Man sieht daraus, dass eine normalisierte Zahl durch eine Mantisse zwischen 0,1 und 1 gekennzeichnet ist, sofern es sich nicht um den Wert Null handelt.
Mit anderen Worten lässt sich dies mathematisch so darstellen:

$$0,1 <= M < 1 \quad \text{oder} \quad 10^{-1} <= M < 10^{0}$$

Im binären Fall nun gilt entsprechend:

$$0,5 <= M < 1 \quad \text{oder} \quad 2^{-1} <= M < 2^{0}$$

Dabei ist M der Absolutbetrag der Mantisse, d.h. ihr Wert ohne Vorzeichen.

Beispiel

101.11 wird normalisiert zu $0.10111*2^{+3}$

In der Darstellung kann das "0." auch noch wegfallen, so dass die Mantisse die Form 10111 und der Exponent den Wert 3 bekommt.
Diese Darstellung wird auch im CBM benutzt mit folgenden Besonderheiten:

a. Beim Exponenten wird von einer Basis von $80 ausgegangen, d.h. der Exponent wird mit $80 addiert angegeben.

   Beispiel $82 = *2
   -------- $7A = *2

b. Die Mantisse wird ohne ihr höchstwertiges Bit abgespeichert. Dieses wird vorher herausgeschoben. An die Stelle des höchstwertigen Bits tritt das Vorzeichen der Mantisse (0 für positiv,1 für negativ).

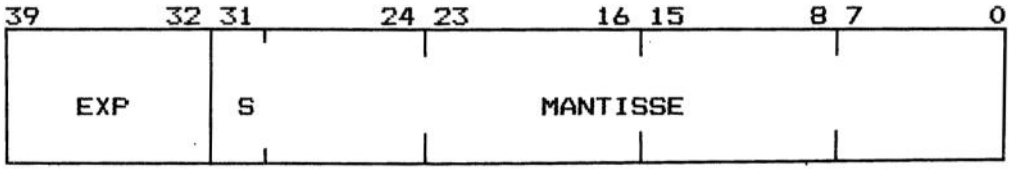
Weiterhin gibt es bei den COMMODORE- Rechnern zwei verschiedene Formate:

a. Speicher-Format
b. Register-Format

Speicher-Format

Dieses Format entspricht der Ablage der Gleitkommavariablen in der Variablentabelle.
Zur Darstellung werden fünf Bytes benutzt. Das erste Byte enthält den Exponenten und Byte zwei bis fünf die Mantisse als vorzeichenbehaftete Zahl.

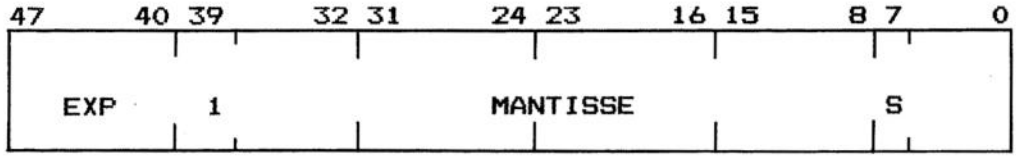
Beachte: Keine Darstellung im 2er-Komplement wie bei negativen ganzen Zahlen.



S = Sign Bit (Vorzeichenbit)

Registerformat

In den Rechenregistern (FAC#1 und FAC#2) werden Fliesskommazahlen in sechs aufeinanderfolgenden Bytes abgebildet.
Die Bytes 1 sowie 3-5 sind identisch zum Speicherformat. Byte 2 des Registerformats entspricht Byte 2 des Speicherformats, allerdings ist hier das höchstwertige Bit immer gesetzt (ausser beim Wert Null). Byte 6 des Registerformats ist identisch zum Byte 2 des Speicherformats und beinhaltet als Information das Vorzeichen der Mantisse.
Wie man sieht, enthalten also die Bytes 2-5 den Absolutbetrag der Mantisse und das Byte 6 gibt an, ob die Zahl positiv oder negativ ist.



Beachte: Das Byte 6 im Registerformat hat im Prinzip nichts mit der eigentlichen Zahlendarstellung zu tun. Es enthält nur das Vorzeichen der Mantisse im höchstwertigen Bit. Die sieben niederwertigen Bits haben keinerlei Bedeutung und können beliebig Einsen oder Nullen enthalten.
Es darf also nicht der Schluss gezogen werden, dass beim Registerformat im Gegensatz zum Speicherformat eine Vergrösserung der Mantisse stattfinden könnte.

Beispiele zur Gegenüberstellung Speicher - Registerformat

Dezimaler Wert	Speicherformat	Registerformat
+ 1	81 00 00 00 00	81 80 00 00 00 00
+ 2.5	82 20 00 00 00	82 A0 00 00 00 20
+ 5	83 20 00 00 00	83 A0 00 00 00 20
+ 20	85 20 00 00 00	85 A0 00 00 00 20
- 1	81 80 00 00 00	81 80 00 00 00 80
- 3	82 C0 00 00 00	82 C0 00 00 00 C0
- 16.5	85 84 00 00 00	85 84 00 00 00 84
- 0.25	7F 80 00 00 00	7F 80 00 00 00 80

Weitere Beispiele der Zahlendarstellung im Speicherformat befinden sich im Anhang 21/22.

5.2 Verwendbare Routinen

Bevor auf die Benutzung von mathematischen Betriebssystemroutinen eingegangen wird, hier noch einige Vorbemerkungen:

a. Sämtliche mathematischen Operationen werden als Fliesskommaoperationen durchgeführt. Das bedingt jedoch, dass Integerwerte vor mathematischen Berechnungen in die äquivalente Fliesskommadarstellung umgewandelt werden müssen. In der Beschreibung der einzelnen Routinen wird noch gesondert darauf hingewiesen.

b. Für Fliesskommaoperationen gibt es zwei Pseudoregister, die jeweils aus sechs Bytes bestehen, und die zu verknüpfenden Zahlen enthalten. Alle Routinen, die nachfolgend besprochen werden, greifen auf eines dieser beiden Register oder auch auf beide zu.
Diese Register sind folgendermassen in der Zeropage angelegt:

Bedeutung	Adresse
Fliesskommaakkumulator 1 (FAC#1)	94-99=$5E-63
Fliesskommaakkumulator 2 (FAC#2)	102-107=$66-$6B

Welches der beiden Rechenregister von der entsprechenden Routine benutzt wird, ist jeweils aus "wichtige Speicherzellen" ersichtlich.

c. Auf eine genaue Analyse der verwendeten Maschinensprachelösungen für die mathematischen Funktionen wird hier verzichtet. Die prinzipiellen Lösungsansätze können in Fachbüchern nachgelesen werden.

A. Umwandlung einer ganzen positiven Zahl in eine Fliesskommazahl — FLOATC 56149 < 52607

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)
95=$5F	HIGH-Byte der ganzen Zahl
96=$60	LOW-Byte der ganzen Zahl

Voraussetzungen

1. Die ganze Zahl muss in 95,96 (HB,LB) bereitgestellt werden.
2. Das X-Register muss mit $90 geladen werden.
3. Das CARRY-Bit muss gesetzt werden.

Ablauf der Routine

Die vorzeichenlose, ganze Zahl (Bereich:0-65535) wird in das Fliesskommaformat umgewandelt und im FAC#1 zur Weiterverarbeitung bereitgestellt.

B. Umwandlung einer ganzen Zahl (positiv und negativ) in eine Fliesskommazahl GIVAYF 53869 < 50364

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Die zu wandelnde ganze Zahl muss im Akku (HIGH-Byte) und im Y-Register (LOW-Byte) übergeben werden.

Ablauf der Routine

Die vorzeichenbehaftete ganze Zahl (Bereich: -32768 bis +32767) wird in die entsprechende Fliesskommazahl konvertiert. Das Ergebnis steht dann im FAC#1.

C. Umwandlung einer Fliesskommazahl in eine ganze Zahl (positiv und negativ)

AYINT 53402<49898

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)
97=$61	HIGH-Byte der ganzen Zahl
98=$62	LOW-Byte der ganzen Zahl

Voraussetzungen

Die zu wandelnde Fliesskommazahl muss in den FAC#1 gebracht werden.

Ablauf der Routine

Nachdem die Fliesskommazahl auf den zulässigen Zahlenbereich (-32768 bis +32767) überprüft ist, wird sie in eine Integerzahl konvertiert. Das Ergebnis steht dann in 97,98 (HB,LB). Bei einer Überschreitung des Zahlenbereichs erfolgt "ILLEGAL QUANTITY ERROR".

D. Umwandlung einer Fliesskommazahl in eine positive ganze Zahl

GETADR 54994 < 51501

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)
17=$11	LOW-Byte der ganzen Zahl
18=$12	HIGH-Byte der ganzen Zahl

Voraussetzungen

Die Fliesskommazahl muss im FAC#1 stehen.

Ablauf der Routine

Nach einer Überprüfung des zulässigen Zahlenbereichs (0 bis 65553) wird die Fliesskommazahl in eine vorzeichenlose ganze Zahl umgewandelt und nach 17,18 (LB,HB) gespeichert.
Bei einer überschreitung des möglichen Zahlenbereichs erfolgt "ILLEGAL QUANTITY ERROR".
Weiterhin steht die Zahl nach dem Rücksprung im Akku (HIGH-Byte) und im Y-Register (LOW-Byte) zur sofortigen Weiterverarbeitung zur Verfügung.

E. Umwandlung eines Strings in eine Fliesskommazahl

STRFKZ 54927 < 51435

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)
31=$1F	Stringadresse LB
32=$20	Stringadresse HB

Voraussetzungen

1. Die Anfangsadresse des Strings muss in 31,32 (LB,HB) übergeben werden.
2. Die Länge des Strings muss beim Aufruf im Akku stehen.

Ablauf der Routine

Der ASCII-String aus Ziffern (evtl. Dezimalpunkt und Exponent) wird in eine Fliesskommazahl umgewandelt. Nach dem Rücksprung steht in FAC#1 der Wert des Strings als Fliesskommazahl.
Wird bei der Umwandlung eine Ueberschreitung des zulässigen Zahlenbereiches entdeckt, so erfolgt "OVERFLOW ERROR".

F. Umwandlung einer Fliesskommazahl in einen String	FOUT	56553 < 53139

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Die zu wandelnde Zahl muss vor dem Aufruf im FAC#1 stehen.

Ablauf der Routine

Nach der Überprüfung auf positiven oder negativen Wert wird die Fliesskommazahl in den entsprechenden ASCII-String umgewandelt.
Der entstandene String beginnt ab $0100, also im Stackbereich des Rechners. Es ist hier also Vorsicht geboten, wenn der Stack durch viele Unterprogrammsprünge bereits nach unten hin sehr gefüllt ist.
Ein Absturz des Systems ist die Folge, wenn der String mit dem augenblicklich benutzten Stackbereich kollidiert.
Bei der Umwandlung wird als Endezeichen ein 00-Byte an den String angehängt und seine Anfangsadresse im AC,YR (LB,HB) übergeben.
Diese Tatsache kann man ausnutzen, den String gleich nach der Umwandlung mit Hilfe einer Betriebssystemroutine (JSR $CA1C) auf dem Bildschirm auszugeben, die diese Einsprungbedingungen benötigt.

Anmerkung: Verwendet man als Einsprungsadresse nicht FOUT, sondern FOUT-6, so wird der Inhalt von FAC#1 nach der Umwandlung in einen String automatisch auf dem Bildschirm ausgedruckt.

Bemerkungen zu den Beispielen

Beispiel zu B

Aufgabenstellung

Es soll die ganze Zahl -32768 in eine Fliesskommazahl umgewandelt werden.

Lösung

Die ganze Zahl, die im Integerformat dem Wert $8000 entspricht, wird im AC und YR übergeben.
Zur Kontrolle wird nach der Konvertierung der Wert von FAC#1 ausgegeben.

Beispiel zu C

Aufgabenstellung

Es soll die Fliesskommazahl +50.0 in eine ganze Zahl umgewandelt werden.

Lösung

Zuerst wird der Wert von FAC#1 auf 50.0 gesetzt:

```
50.0 = 86 C8 00 00 00 48
```

Nach dem Rücksprung aus der Umwandlungsroutine kann man sich die entsprechende ganze Zahl im Monitor in $61,62 (HB,LB) ansehen.

Beispiel zu E

Aufgabenstellung

Es soll der ASCII-String -1.2345, der ab $034E im Speicher steht, in eine Fliesskommazahl konvertiert werden.

Lösung

Der ASCII-String wird ab $034E abgelegt, im Akku wird seine Länge (=7) übergeben und die Umwandlungsroutine aufgerufen.
Zur Kontrolle des Ergebnisses wird noch der Inhalt von FAC#1 auf dem Bildschirm ausgegeben.

```
                    0001              ;BEISPIEL ZU B.:
                    0002              ;--------------------------------
                    0003 GIVAYF       .DE $D26D (<C4BC) ;WANDELT INTEGERZAHL
                    0004              ;                  UM IN FLIESSKOMMAZAHL
                    0005 LEERZEILE    .DE $E52F (<E3B6)
                    0006 FOUT/6       .DE $DCE3 (<CF8D)
                    0007              ;--------------------------------
                    0008              .BA $033A
033A- A9 80         0009 ANFANG       LDA #$80      ;HB DER INTEGERZAHL
033C- A0 00         0010              LDY #$00      ;LB DER INTEGERZAHL
033E- 20 6D D2      0011              JSR GIVAYF
0341- 20 2F E5      0012              JSR LEERZEILE
0344- 20 E3 DC      0013              JSR FOUT/6    ;GIBT INHALT VON FAC#1
                    0014              ;              AUF DEN BILDSCHIRM AUS
0347- 00            0015              BRK
                    0016              .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/GIVAYF=D26D (<C4BC)          /LEERZEILE=E53F (<E3B6)
/FOUT/6=DCE3 (<CFD8)           ANFANG=033A
```

```
                    0001            ;BEISPIEL ZU C.:
                    0002            ;---------------------------------
                    0003 AYINT      .DE $D09A (<C2EA) ;WANDELT FLIESSKOMMAZAHL
                    IM
                    0004            ;                       FAC#1 UM IN INTEGERZAHL
                    0005 FAC1       .DE $5E
                    0006            ;
                    0007            ;---------------------------------
                    0008            ;
                    0009            .BA $033A
033A- A9 00         0010 ANFANG     LDA #$00
033C- AA            0011            TAX           ;XR LOESCHEN
033D- BD 4B 03      0012 HOLEN      LDA TABELLE,X
0340- 95 5E         0013            STA *FAC1,X   ;FAC#1 MIT 50.0 LADEN
0342- E8            0014            INX
0343- E0 06         0015            CPX #$06      ;6.ZEICHEN ?
0345- D0 F6         0016            BNE HOLEN     ;NEIN:=>NAECHSTES BYTE
0347- 20 9A D0      0017            JSR AYINT
034A- 00            0018            BRK
034B- 86 C8 00      0019 TABELLE    .BY $86 $C8 $00 $00 $00 $48
034E- 00 00 48                      ; = 50.0
                    0020            .EN
```

```
LABEL FILE:  ( / = EXTERNAL )

/AYINT=D09A (<C2EA)     /FAC1=005E              ANFANG=033A
HOLEN=033D               TABELLE=034B
```

```
                    0001              ;BEISPIEL ZU E.:
                    0002              ;------------------------------
                    0003 STRINGADR    .DE $1F       ;POINTER AUF DEN
                    0004              ;              ANFANG DES STRINGS
                    0005 STRFKZ       .DE $D68F (<CBEB) ;WANDELT STRING UM
                    0006              ;                 IN FLIESSKOMMAZAHL
                    0007 LEERZEILE    .DE $E52F (<E3B6)
                    0008 FOUT/6       .DE $DCE3 (<CF8D) ;GIBT DEN INHALT
                    0009              ;                 VON FAC#1 AUS
                    0010              ;------------------------------
                    0011              .BA $033A
033A- A9 4E         0012 ANFANG       LDA #L,STRING          ;POINTER AUF
033C- 85 1F         0013              STA *STRINGADR         ;DEN ANFANG
033E- A9 03         0014              LDA #H,STRING          ;DES STRINGS
0340- 85 20         0015              STA *STRINGADR+1       ;SETZEN
0342- A9 07         0016              LDA #7        ;LAENGE DES STRINGS
0344- 20 8F D6      0017              JSR STRFKZ
0347- 20 2F E5      0018              JSR LEERZEILE
034A- 20 E3 DC      0019              JSR FOUT/6
034D- 00            0020              BRK
034E- 2D 31 2E      0021 STRING       .BY '-1.2345'
0351- 32 33 34
0354- 35
                    0022              .EN

LABEL FILE:   ( / = EXTERNAL )

/STRINGADR=001F          /STRFKZ=D68F (<C8EB)     /LEERZEILE=E52F (<E3B6)
/FOUT/6=DCE3 (<CF8D)      ANFANG=033A              STRING=034E
```

G. Übertragung von FAC#1 nach FAC#2 MOVEF 56091 < 52551

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E=63	Fliesskommaakkumulator 1 (FAC#1)
102-107=$66-$6B	Fliesskommaakkumulator 2 (FAC#2)

Voraussetzungen

Keine

Ablauf der Routine

Der Inhalt von FAC#1 wird nach FAC#2 kopiert.
Der Wert von FAC#1 steht dann in FAC#1 und FAC#2 zur Verfügung.
Soll der Wert von FAC#1 vorher gerundet werden, so ist als Einsprungadresse $DB18 zu wählen.

H. Übertragung von FAC#2 nach FAC#1 MOVFA 56072 < 52530

Wichtige Speicherzellen

siehe G.

Voraussetzungen

Keine

Ablauf der Routine

Der Inhalt von FAC#2 wird nach FAC#1 kopiert.
Der Wert von FAC#2 steht dann sowohl im FAC#2 als auch im FAC#1 zur Verfügung.

I. Übertragung von FAC#1 nach Zwischenspeicher 1

MOV1F 56022 < 52480

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)
84-88=$54-58	Zwischenspeicher 1

Voraussetzungen

Keine

Ablauf der Routine

Der Inhalt von FAC#1 wird in den Zwischenspeicher 1 transferiert.
Der Wert von FAC#1 liegt dann im Zwischenspeicher 1 im Speicherformat vor.

J. Übertragung von FAC#1 nach Zwischenspeicher 2

MOV2F 56019 < 52477

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5e-63	Fliesskommaakkumulator 1 (FAC#1)
89-93=$59-5D	Zwischenspeicher 2

Voraussetzungen

Keine

Ablauf der Routine

Der Inhalt von FAC#1 wird in den Zwischenspeicher 2 transferiert mit gleichzeitiger Umwandlung in das Speicherformat. Diese und die vorhergehende Routine kann man benutzen, um Zwischenergebnisse aus mathematischen Berechnungen, die noch benötigt werden, in einen der beiden Zwischenspeicher zu retten.

K. Laden von FAC#1 mit einer Konstanten	MOVFM	55982 < 52440

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Die Anfangsadresse der Konstanten muss vor dem Aufruf im AC,YR (LB,HB) übergeben werden.
Die Konstante muss im Speicherformat vorliegen.
Sie kann z.B. auch der Inhalt einer Gleitkommavariablen während eines Programmlaufes sein.

Ablauf der Routine

Die Routine transportiert die Konstante aus dem Speicher in den FAC#1, wobei sie gleichzeitig das Speicherformat in das Registerformat umwandelt.

L. Laden von FAC#2 mit einer Konstanten	CONUPK	55704 < 52162

Wichtige Speicherzellen

Adresse	Bedeutung
102-107=$66-6B	Fliesskommaakkumulator 2 (FAC#2)

Voraussetzungen

Siehe K.

Ablauf der Routine

Die Routine transportiert die Konstante aus dem Speicher in den FAC#2 mit gleichzeitiger Umwandlung vom Register- in das Speicherformat.

Hinweis zu K und L

Im Betriebssystem ist eine Reihe von Gleitkommakonstanten enthalten, die auch für die eigene Assemblerprogrammierung verwendet werden können und deren Anfangsadressen nachstehend aufgeführt sind:

Anfangsadresse		Wert der Konstanten
BASIC 3.0	BASIC 4.0	
57438=$E05E	54024=$D308	+ 0.25
56861=$DE1D	53447=$D0C7	+ 0.5
55532=$D8EC	51990=$CB16	- 0.5
55496=$D8C8	53631=$D17F	+ 1.0
55813=$DA05	52271=$CC2F	+ 10
56511=$DCBF	52969=$CEE9	+ 99999999.9
56516=$DCC4	52974=$CEEE	+ 999999999
56521=$DCC9	52979=$CEF3	+ 1E+09

Weitere wichtige Konstanten:

57428=$E054	54014=$D2FE	+ PI/2
57433=$E059	54055=$D327	+ 2*PI
52643=$CDA3		+ PI
55502=$D8CE	51960=$CAF8	LOG(e)= .4342
(Basis 10)		
55537=$D8F1	51995=$CB1B	LN(2) = .6931
55527=$D8E7	51985=$CB12	SQR(2)=1.414
55522=$D8E2	51980=$CB0C	1/SQR(2)= .0707

M. Übertragung von FAC#1 nach Speicher	MOVMF	56035 < 56586

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Die Anfangsadresse, ab der die Gleitkommazahl aus dem FAC#1 im Speicher abgelegt werden soll, muss vor dem Aufruf der Routine in XR,YR (LB,HB) übergeben werden.

Ablauf der Routine

Die Routine transportiert die Gleitkommazahl aus dem FAC#1 an die gewünschte Stelle im RAM mit gleichzeitiger Umwandlung vom Registerformat in das Speicherformat.
Soll vor dem Transport der Wert von FAC#1 gerundet werden, so ist als Einsprungadresse MOVMF-3 zu nehmen.

Bemerkungen zum Beispiel von K

Zugehöriges BASIC-Programm

```
10 A=9.87654321
20 END
```

Aufgabenstellung

Es soll der Inhalt der Variablen A in den FAC#1 übertragen werden.

Lösung

Zuerst wird die Anfangsadresse des Variableninhaltes von A gesucht.(Siehe Speicherverwaltung:Routine G)
Diese Adresse wird im AC,YR der Laderoutine übergeben.
Nach dem Rücksprung befindet sich dann in FAC#1 der Wert der Variablen A.
Zur Kontrolle wird der Inhalt von FAC#1 auf dem Bildschirm ausgegeben.

```
                0001            ;BEISPIEL ZU K.:
                0002            ;------------------------------
                0003 VARNAME    .DE $42    ;NAME DER VARIABLEN
                0004 VARADRES   .DE $44    ;POINTER AUF DIE
                0005            ;           ANFANGSADRESSE
                0006            ;           DER VARIABLEN
                0006 SUCHEN     .DE $CFC9 ;SUCHT ANFANGSADRESSE
                0007                (<C187)      ;DER VARIABLEN
                0008 MOVFM      .DE $DAAE ;UEBERTRAEGT VARIABLE
                0009            ;     (CCD8)       ;NACH FAC#1
                0010 LEERZEILE  .DE $E52F  (<E3B6)
                0011 AUSGABE    .DE $DCE3 ;GIBT DEN INHALT
                0012            ;     (CF8D) ;DES FAC#1 AUS
                0013            ;------------------------------
                0014            .BA $033A
033A- A9 41     0015 ANFANG     LDA #'A          ;NAMEN DER
033C- 85 42     0016            STA *VARNAME     ;VARIABLEN
033E- A9 00     0017            LDA #0           ;ABLEGEN
0340- 85 43     0018            STA *VARNAME+1
0342- 20 C9 CF  0019            JSR SUCHEN
0345- A5 44     0020            LDA *VARADRES    ;ANFANGS-
                0021            ;                 ADRESSE VON
0347- A4 45     0021            LDY *VARADRES+1  ;A UEBERGEBEN
0349- 20 AE DA  0022            JSR MOVFM
034C- 20 2F E5  0023            JSR LEERZEILE
034F- 20 E3 DC  0024            JSR AUSGABE
0352- 00        0025            BRK
                0026            .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/VARNAME=0042          /VARADRES=0044          /SUCHEN=CFC9 (<C187)
/MOVFM=DAAE (<CCD8)    /LEERZEILE=E52F (<E3B6) /AUSGABE=DCE3 (<CF8D)
ANFANG=033A
```

N. Arithmetische Operationen zwischen den Rechenregistern

Wichtige Speicherzellen

Adresse	Bedeutung
94-99 =$5E-63	Fliesskommaakkumulator 1 (FAC#1)
102-107=$66-6B	Fliesskommaakkumulator 2 (FAC#2)

Voraussetzungen

Vor dem Aufruf der entsprechenden Routine müssen die zu verknüpfenden Fliesskommazahlen im FAC#1 und FAC#2 bereitgestellt werden.
Hierbei ist zu beachten, dass sämtliche Funktionen als Fliesskomma-Operationen durchgeführt werden. Dies bedingt, dass vor der Verknüpfung von Integerzahlen mit Integer- oder Gleitkommazahlen eine Umwandlung des Integer- in das Fliesskommaformat erfolgt. Entsprechend muss auch wieder eine Rücktransformation stattfinden, wenn das Ergebnis als Integerzahl verarbeitet werden soll.

Folgende Routinen stellt das Betriebssystem zur Verfügung:

Math. Operation	Einsprungadresse		LABEL
	BASIC 3.0	BASIC 4.0	
FAC#1=-(FAC#1)	56993=$DEA1	53579=$D14B	NEGOP
FAC#1=(FAC#1) + 0.5	55084=$D72C	51583=$C07F	FADDH
FAC#1=(FAC#1) * 2.0	55808=$DA00	52266=$CC2A	MUL02
FAC#1=(FAC#1) * 10.0	55790=$D9EE	52248=$CC18	MUL10
FAC#1=(FAC#1) / 10.0	55818=$DA0A	52276=$CC34	DIV10
FAC#1=(FAC#2) + (FAC#1	55163=$D77B	51621=$C9A5	ADFACS
FAC#1=(FAC#2) - (FAC#1	55094=$D736	51593=$C989	FSUBT
FAC#1=(FAC#2) * (FAC#1	55612=$D93C	52070=$CB66	MULFAC
FAC#1=(FAC#2) / (FAC#1	55840=$DA20	52296=$CC48	FDIVT
FAC#1=(FAC#2)hoch(FAC#1)	56938=$DE6A	53522=$D112	FPWRT

Ablauf der Routinen

Nach Berechnung der gewünschten Verknüpfung steht das Ergebnis im FAC#1 zur Verfügung.

0. Arithmetische Operationen zwischen einem Rechenregister und einer Konstanten

Wichtige Speicherzellen

Adresse	Bedeutung
94-99 =$5E-63	Fliesskommaakkumulator 1 (FAC#1)
102-107=$66-6B	Fliesskommaakkumulator 2 (FAC#2)

Voraussetzungen

Der eine Operand muss im FAC#1 bzw. FAC#2 bereitgestellt werden.
Die Anfangsadresse der Konstanten muss im AC,YR (LB,HB) der aufgerufenen Routine übergeben werden.
Die Konstante muss im Speicherformat angegeben sein.
Sie kann z.B. der Inhalt einer Gleitkommavariablen während einem Programmlauf sein, deren Anfangsadresse mit Hilfe von Betriebssystemroutinen gefunden werden kann (siehe Speicherverwaltung), oder auch eine der Konstanten, die im ROM-Bereich abgelegt sind (siehe unter L).

Folgende Routinen besitzt das Betriebssystem:

Math. Operation	Einsprungadresse		LABEL
	BASIC 3.0	BASIC 4.0	
FAC#1=Konst. + (FAC#1)	55155=$D773	51613=$C99D	FADD
FAC#1=Konst. - (FAC#1)	55091=$D733	51590=$C986	FSUB
FAC#1=Konst. * (FAC#1)	55604=$D934	52062=$CB5E	FMULT
FAC#1=Konst. / (FAC#1)	55835=$DA1B	52293=$CC45	FDIV
FAC#1=(FAC#2)hoch Konst.	56933=$DE65	53519=$D10F	FEXP

Ablauf der Routinen

Die Routine transferiert zuerst die Konstante in das entsprechende Rechenregister mit gleichzeitiger Umwandlung des Abspeicherformates und führt dann die gewünschte Berechnung durch.
Das Ergebnis steht nach dem Rücksprung im FAC#1 zur Weiterverarbeitung zur Verfügung.

P. Monadische Operationen

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Das Argument muss im FAC#1 bereitgestellt werden.

Mögliche Funktionen

Math. Operation	Einsprungadresse BASIC 3.0	Einsprungadresse BASIC 4.0	LABEL
FAC#1=SGN(FAC#1)	56133=$DB45	52591=$CD6F	SGN
FAC#1=INT(FAC#1)	56280=$DBD8	52738=$CE02	INT
FAC#1=ABS(FAC#1)	56164=$DB64	52622=$CD8E	ABS
FAC#1=SQR(FAC#1)	56926=$DE5E	53512=$D108	SQR
FAC#1=LOG(FAC#1)	55542=$D8F6	52000=$CB20	LOG
FAC#1=EXP(FAC#1)	57050=$DEDA	53636=$D184	EXP
FAC#1=SIN(FAC#1)	57311=$DFDF	53897=$D289	SIN
FAC#1=COS(FAC#1)	57304=$DFD8	53890=$D282	COS
FAC#1=TAN(FAC#1)	57384=$E028	53970=$D2D2	TAN
FAC#1=ATN(FAC#1)	57484=$E08C	54060=$D32C	ATN

Ablauf der Routinen

Nach Berechnung der mathematischen Funktion steht das Ergebnis im FAC#1 zur Verfügung.

Q. Zufallszahl im FAC#1 erzeugen — RND — 57215 < 53801

Wichtige Speicherzellen

Adresse	Bedeutung
94-99 =$5E-63	Fliesskommaakkumulator 1 (FAC#1)
136-140=$88-8C	Zufallszahl

Voraussetzungen

Keine

Ablauf der Routine

Die augenblickliche Zufallszahl wird in den FAC#1 gebracht. Dort wird sie nach einem vorgegebenen Algorithmus verändert und als neue Zufallszahl wieder nach $88-8C übertragen. Nach dem Rücksprung steht sie auch noch im FAC#1 zur Weiterverarbeitung bereit.

R. Prüfung von FAC#1 auf Vorzeichen und 0	SIGN	56119 < 52577

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

Die zu prüfende Zahl muss beim Aufruf der Routine im FAC#1 stehen.

Ablauf der Routine

Die Speicherzelle ($5E), die den Exponenten enthält, wird auf den Wert Null geprüft. Sie ist dann gleich Null, wenn sie den Wert 2 enthält:
Mantisse * 2 entspricht aufgrund der Zahlendarstellung dem Wert Null.
Die Prüfung auf positiven oder negativen Inhalt wird am Byte 6 von FAC#1 vorgenommen, das im höchstwertigen Bit das Vorzeichen der Fliesskommazahl enthält.
Nach dem Rücksprung enthalten bestimmte Flags des Statusregisters und der Inhalt des Akku's das Ergebnis des Tests, was aus nachfolgender Aufstellung hervorgeht:

FAC#1	AC	C-Bit	Z-Bit	N-Bit
> 0	1	0	0	0
= 0	0	*	1	0
< 0	$FF	1	0	1

S. Vergleich von FAC#1 mit einer Konstanten	FCOMP	56167 < 52625

Wichtige Speicherzellen

Adresse	Bedeutung
94-99=$5E-63	Fliesskommaakkumulator 1 (FAC#1)

Voraussetzungen

FAC#1 muss die Zahl enthalten, die verglichen werden soll.
Die Anfangsadresse der Vergleichszahl muss im AC,YR (LB,HB) übergeben werden.
Die Vergleichszahl muss im Speicherformat vorliegen.

Ablauf der Routine

Zuerst wird überprüft, ob die beiden Zahlen unterschiedliche Vorzeichen besitzen.
Danach werden die einzelnen Stellen der Zahlenregister miteinander verglichen.
Je nach Ausgang des Vergleichs werden einzelne Flags im Statusregister gesetzt bzw. gelöscht.
Nach dem Rücksprung kann man dann an diesen Flags und auch am Wert im Akku das Testergebnis ablesen:

FAC#1	AC	C-Bit	Z-Bit	N-Bit
> Konst.	1	0	0	0
= Konst.	0	*	1	0
< Konst.	$FF	1	0	1

Ein abschliessendes komplexeres Beispiel soll die Verwendung der mathematischen Routinen verdeutlichen.
Um dieses Assemblerprogramm zu verstehen, muss man sich die Speicherverwaltung ansehen.

Aufgabenstellung

Es sei folgendes BASIC-Programm gegeben::

```
10 A=9
20 B=4
30 SYS 826
40 PRINT "C=SQR(";A;"*";B;")=";C
50 END
```

In dem Maschinenprogramm, das durch SYS826 aufgerufen wird, soll nun folgende mathematische Berechnung durchgeführt werden:

$$C = \sqrt{A*B}$$

Das Ergebnis soll nach der Berechnung in der Variablen C stehen.

Lösung

Zuerst wird die Anfangsadresse des Variableninhaltes von A und B gesucht (siehe Speicherverwaltung: Routine G).
Die Werte der beiden Variablen A bzw. B werden dann in den FAC#1 bzw. FAC#2 übertragen (Routine K bzw. L).
Nach Berechnung von A*B mit Routine N wird mit Routine P die Wurzel aus diesem Ergebnis gezogen. Der Wert der gewünschten Berechnung steht nun im FAC#1.
Um dieses Ergebnis in die Variable C zu übertragen, wird wiederum die Variablensuchroutine aufgerufen, die eine neue Variable C anlegt, da sie ja noch nicht existiert.
Der Inhalt von FAC#1 kann nun mit Routine M in die Variable C transferiert werden.
Nach dem Rücksprung zu BASIC wird der Wert der Variablen C ausgegeben (in diesem Fall die Zahl 6).

```
                    0000                        ;Abschliessendes Beispiel
                                                ;zur Mathematik
                                               ;--------------------------------
                    0001 NAME                  .DE $42     ;NAME DER VARIABLEN
                    0002 VARADR                .DE $44     ;VARIABLENADRESSE
                    0003 VARSUCHE              .DE $CFC9   ;SUCHT ANFANGSADRESSE
                    0004                       ;DES VARIABLENINHALTES
                    0005 MOVFM                 .DE $DAAE (<CCD8) ;VARIABLE NACH FAC#1
                    0006 CONUPK                .DE $D998 (<CBC2) ;VARIABLE NACH FAC#2
                    0007 MOVMF                 .DE $DAE0 (<CD0D) ;FAC#1 NACH VARIABLE
                    0008 MULFAC                .DE $D93C (<CB66) ;FAC#1=FAC#1*FAC#2
                    0009 SQR                   .DE $DE5E (<D108) ;FAC#1=SQR(FAC#1)
                    0010                       ;--------------------------------
                    0011                       ;
                    0012                       .BA $033A
033A- A9 41         0013 ANFANG                LDA #'A          ;VARIABLENNAME
033C- 85 42         0014                       STA *NAME        ; ="A" FESTLEGEN
033E- A9 00         0015                       LDA #00
0340- 85 43         0016                       STA *NAME+1
0342- 20 C9 CF      0017                       JSR VARSUCHE
0345- A5 44         0018                       LDA *VARADR  ;VARIABLENADRESSE IN
0347- A4 45         0019                       LDY *VARADR+1  ;AC UND YR UEBERGEBEN
0349- 20 AE DA      0020                       JSR MOVFM
034C- A9 42         0021                       LDA #'B          ;VARIABLENNAME
034E- 85 42         0022                       STA *NAME        ; ="B" FESTLEGEN
0350- A9 00         0023                       LDA #00
0352- 85 43         0024                       STA *NAME+1
0354- 20 C9 CF      0025                       JSR VARSUCHE
0357- A5 44         0026                       LDA *VARADR  ;VARIABLENADRESSE IN
0359- A4 45         0027                       LDY *VARADR+1 ;AC UND YR UEBERGEBEN
035B- 20 98 D9      0028                       JSR CONUPK
035E- 20 3C D9      0029                       JSR MULFAC
0361- 20 5E DE      0030                       JSR SQR
0364- A9 43         0031                       LDA #'C          ;VARIABLENNAME
0366- 85 42         0032                       STA *NAME        ; ="C" FESTLEGEN
0368- A9 00         0033                       LDA #00
036A- 85 43         0034                       STA *NAME+1
036C- 20 C9 CF      0035                       JSR VARSUCHE ;VARIABLE C NEU ANLEGEN
036F- A6 44         0036                       LDX *VARADR  ;VARIABLENADRESSE IN
0371- A4 45         0037                       LDY *VARADR+1 ;XR UND YR UEBERGEBEN
0373- 20 E0 DA      0038                       JSR MOVMF
0376- 60            0039                       RTS
                    0040                       .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/NAME=0042              /VARADR=0044            /VARSUCHE=CFC9
/MOVFM=DAAE (<CCD8)     /CONUPK=D998 (<CBC2)    /MOVMF=DAE3 (<CD0D)
/MULFAC=D93C (<CB66)    /SQR=DE5E (<D108)        ANFANG=033A
```

6. BASIC-Interpretation

Das Rechnersystem CBM besitzt einen 8 k-Byte BASIC-Interpreter, der von der Firma MICROSOFT entwickelt wurde.

In diesem Kapitel soll nun die Arbeitsweise des BASIC-Interpreters untersucht werden, d.h. es soll die Frage beantwortet werden, wie der BASIC-Interpreter die Ausführung eines BASIC-Programmes bzw. eines BASIC-Direktkommandos erreicht und welche Voraussetzungen er dafür benötigt.

6.1 Prinzipielle Vorgehensweise

Solange kein BASIC-Programm oder Direktkommando bearbeitet wird, befindet sich das System in einer Eingabe-Warteschleife.
Werden Zeichen über die Tastatur eingegeben, so werden sie zunächst hintereinander im Tastaturpuffer zwischengespeichert. In dieser Warteschleife wird dann der Tastaturpuffer byteweise abgebaut, wobei die einzelnen Zeichen auf dem Bildschirm dargestellt und danach im BASIC-Input-Puffer (512-592) abgelegt werden.
Dies wird solange fortgesetzt, bis ein CARRIAGE RETURN auftritt.
Das Betriebssystem verlässt daraufhin die Eingabe-Warteschleife und überprüft, ob es sich bei dem eingegebenen Text um ein Direktkommando oder eine Programmzeile handelt:
Wenn das 1.Zeichen im Input-Puffer eine Ziffer ist, so wird eine Programmzeile erkannt, ansonsten ein Direktkommando.

Dementsprechend verläuft die weitere Bearbeitung der Eingabe:

a. Bei einer Programmzeile wird zuerst die Zeilennummer, die ja im ASCII-Format vorliegt, in den entsprechenden 2-Byte Code umgewandelt, in dem sie im Programmspeicher abgebildet wird.
   Danach wird der eigentliche Programmtext im Input-Puffer in den BASIC-Kode umgewandelt und nach Berechnung der Startadresse der neuen Zeile in den Programmspeicher übertragen.
   Muss die Zeile in ein bereits bestehendes Programm eingefügt werden, so wird zuvor noch der nötige Platz im Programmspeicher durch Verschieben der bestehenden Programmzeilen geschaffen.
   Nach Korrektur aller Pointer der Speicherverwaltung, die durch das Anhängen bzw. Einfügen der neuen BASIC-Zeile falsche Werte erhalten haben, wartet das System in der Eingabeschleife wieder auf neue Befehle.

b. Bei einem BASIC-Direktkommando wird der Text aus dem Input-Puffer zunächst ebenfalls in den BASIC-Kode umgewandelt.
Danach jedoch wird -im Gegensatz zu einer Programmzeile- sofort zur Interpreter-Schleife verzweigt, in welcher der jeweilige Befehl identifiziert und zur Ausführung gebracht wird.
Nach Abarbeiten aller eingegebenen Kommandos kehrt das System zur Eingabe-Warteschleife zurück.

6.2 Umwandlung in den BASIC-Kode

Alle BASIC-Befehle werden vor ihrer Ausführung bzw. vor ihrer Ablage als BASIC-Zeile im Programmspeicher in einen Zwischenkode, den sogenannten BASIC-Kode, umgewandelt.
Dabei wird sowohl jeder primäre BASIC-Befehl (PRINT, INPUT, GET, DIM, READ, DATA etc.) als auch alle sekundären Anweisungen (SIN, COS, TAN, RND, LOG, EXP, VAL, +, -, *, /, MID$, LEFT$, etc.) in nur einem Byte verschlüsselt (siehe Anhang 24).
Dies hat nicht nur den Vorteil, dass viel Speicherplatz gespart wird, den die einzelnen BASIC-Befehle in ihrer reinen ASCII-Darstellung benötigen würden, sondern zudem wird die Identifikation eines BASIC-Befehls in der Interpretations- und Ausführungsphase erheblich beschleunigt.

Man spricht deshalb auch von einem sogenannten halbcompilierenden BASIC-Interpreter.

In der Umwandlungsroutine wird nun der gesamte eingegebene BASIC-Text in einer Schleife zeichenweise aus dem Input-Puffer geholt, analysiert und nach der Verschlüsselung wieder in den BASIC-Input-Puffer zurückgespeichert.

Nach Beendigung der Umwandlung befinden sich im Eingabepuffer die Befehle in dem für den Interpreter lesbaren Kode.

Die Verschlüsselung des Textes vom Eingabepuffer verläuft folgendermassen:

1. Das erste Zeichen des Puffers wird geholt und überprüft, ob sein Kode kleiner als 128 ist, da im Graphik-Modus der mögliche Zeichensatz zur Darstellung eines BASIC-Befehls im ASCII-Format bei 127 endet.
   Alle Zeichen mit einem Kode grösser als 128 sind geshiftete Zeichen (Graphikzeichen) und können daher keinen sinnvollen BASIC-Befehl ergeben.
   Eine Ausnahme bildet die Zahl PI, die den Kode 255 besitzt und wieder an ihre ursprüngliche Stelle im Input-Puffer zurückgespeichert wird.
   Alle anderen geshifteten Zeichen werden ignoriert. Es ist daher auch möglich, vor einem BASIC-Befehl beliebige geshiftete Zeichen einzugeben, ohne dass in der Ausführungsphase ein "SYNTAX ERROR" gemeldet wird.
   Diese Abfrage auf geshiftete Zeichen hat jedoch folgenden unangenehmen Nebeneffekt:
   Im Gross/Klein-Modus werden geshiftete Zeichen - die hier ja sinnvoll wären - ignoriert. Das bedeutet nun, dass beispielsweise "LIST" nicht als BASIC-Befehl "list" erkannt wird, wenn die Eingabe in Grossbuchstaben erfolgt.

2. Das erste nicht geshiftete Zeichen, dass gefunden wird, wird auf folgende Möglichkeiten hin untersucht:

   a. Blank (ASCII-Kode:32)
      Dieses Zeichen wird wieder an die gleiche Stelle im Input-Puffer zurückgelegt.

   b. Anführungszeichen (ASCII-Kode:34)
      Das erste Anführungszeichen und alle darauffolgenden Zeichen bis einschliesslich dem nächsten Anführungszeichen (Schlusszeichen) werden im ASCII-Kode wieder an ihren ursprünglichen Platz im Eingabepuffer zurückgelegt.

   c. Fragezeichen = "?" (ASCII-Kode 63)
      Da das Fragezeichen eine Abkürzung des BASIC-Befehls PRINT darstellt, wird anstelle von "?" der BASIC-Kode für PRINT (=153) im Input-Puffer ablegt.

   d. Zahlen von 0-9, Strichpunkt und Doppelpunkt (ASCII-Kode:48-59)
      Diese Zeichen werden im ASCII-Format wieder an die gleiche Stelle im Input-Puffer zurückgespeichert.

e. Restlicher Zeichensatz=
Buchstaben und Verknüpfungszeichen wie +, -, *, /, etc.
(ASCII-Kode:35-47 und 60-127)
Mit diesem Zeichensatz können alle sinnvollen BASIC-Befehle gebildet werden.
Um nun herauszufinden, ob es sich um einen zulässigen Befehl handelt, ist im ROM-Bereich eine Tabelle aller möglichen BASIC-Befehle abgelegt.
In dieser Tabelle (49298-49553 bzw.< 45234-45579) sind die BASIC-Befehle in ihrer ASCII-Darstellung aufgelistet, wobei das Bit7 des letzten Buchstabens eines jeden Befehles zusätzlich gesetzt ist.
Der Interpreter sucht nun die gesamte Befehlstabelle nach der vorliegenden Zeichenkette durch. Dabei führt er einen Wortzähler mit, der bei Null beginnt und mit jedem neu zu testenden Tabellenbefehl um eins erhöht wird.
Dadurch, dass bei erlaubten BASIC-Befehlen das Bit 7 des letzten Zeichens gesetzt ist, wird das Ende eines Befehles markiert und zudem der Suchvorgang erheblich beschleunigt.
Bei der Suche wird das erste Zeichen (ASCII-Kode) aus der eingegebenen Zeichenkette, die einen möglichen Befehl darstellen könnte, hergenommen. Davon wird der erste Zeichenkode des gerade getesteten BASIC-Befehls aus der Tabelle subtrahiert. Ergibt sich dabei der Wert 0, so stimmen die ersten beiden Zeichen überein. Es werden dann die darauf folgenden Zeichen miteinander verglichen, wobei jedesmal erst das nächste Zeichen aus dem Puffer geholt wird.
Ergibt nun ein Ergebnis der weiteren Vergleiche nicht den Wert 0, sondern 128, so ist das Befehlswort gefunden, da ja in der Befehlstabelle das höchstwertige Bit beim letzten Zeichen jedes Befehls gesetzt ist. Der entsprechende BASIC-Kode berechnet sich dann aus dem Wert des Wortzählers+128. Dieser Kode wird im Input-Puffer anstelle der ASCII-Zeichenfolge abgelegt.
Ist das Ergebnis der ersten Subtraktion nicht =0 bzw. bei einer Uebereinstimmung der ersten Zeichen das Ergebnis einer weiteren Subtraktion auch nicht =128, so wird der Wortzähler inkrementiert und das nächste Befehlswort aus der Tabelle mit derselben Zeichenkette verglichen.
Dies wird solange fortgesetzt, bis das Ende der Tabelle erreicht ist, das mit einer Null gekennzeichnet ist.
Hat der Interpreter bis dahin keine Uebereinstimmung erzielen können, so erkennt er, dass es sich bei der getesteten Zeichenfolge um keinen erlaubten BASIC-Befehl handelt. Er legt dann das erste Zeichen dieser Zeichenkette im ASCII-Kode ab.
Damit wird erreicht, dass nach einem erkannten Befehl wie z.B. SPC(auch die dazugehörigen Parameter (hier: Argument und Klammern) abgespeichert werden.
Nach einer erfolglosen Suche in der Tabelle wird das nächste Zeichen (=zweites Zeichen der getesteten Zeichenfolge) aus dem Input-Puffer geholt und zum Schleifenanfang zurückgekehrt.

Ausserdem werden bestimmte Flags gesetzt, wenn die beiden Befehle DATA oder REM auftreten.
Diese Flags veranlassen den Interpreter, alle Zeichen bis zum nächsten Trennzeichen (Doppelpunkt oder 0=Ende der Eingabe) in den Eingabepuffer zurückzuspeichern, ohne Ueberprüfung auf einen möglichen BASIC-Befehl.

3. Nach Untersuchung eines Zeichens auf die in 2. beschriebenen Möglichkeiten wird nach den dazugehörigen Behandlungen stets zum Schleifenanfang (siehe 1.) zurückgekehrt.

4. Um das Ende des Textes im Eingabepuffer zu erkennen, wird bereits in der Eingabewarteschleife nach Auffinden von CR ($0D) eine Null an den Eingabetext angehängt.
Diese Null veranlasst das Beendigen der Umwandlung.

Aus der Beschreibung des Suchvorgangs kann man bereits ersehen, dass nach einer Uebereinstimmung des ersten Zeichens eines Befehlswortes mit dem ersten Zeichen der gerade untersuchten Zeichenfolge nicht mehr überprüft wird, ob die Länge des Befehlswortes aus der Tabelle mit der Länge des eingegebenen Befehls übereinstimmt.
Der Interpreter betrachtet den Befehl als gefunden, wenn eine weitere Subtraktion das Ergebnis 128 liefert.
Diese Tatsache ermöglicht es nun, die BASIC-Befehle bei der Eingabe abzukürzen, indem von dem jeweiligen Befehl nur der erste Buchstabe und der zweite Buchstabe geshiftet eingegeben wird.
Wird der gewünschte Befehl aus der Tabelle getestet, so ergibt sich bei der Subtraktion das erste Mal der Wert Null und beim zweiten Mal der Wert 128. Damit glaubt der Interpreter, den Befehl gefunden zu haben und legt den zugehörigen BASIC-Kode ab.
Zu beachten ist die Reaktion des Interpreters auf diese Eingabemethode bei Befehlen, deren ersten beiden Zeichen identisch sind (z.B. LET und LEN, GOTO und GOSUB oder RETURN und REM):
Der Interpreter erreicht dann nur jeweils den ersten Befehl, welcher dem Vergleich entspricht (=der Befehl, der in der Tabelle weiter vorne steht).
Will man dagegen bei diesen Ausnahmen sichergehen, so muss man die ersten 2 Zeichen eingeben, gefolgt vom geshifteten 3. Zeichen. Damit ist eine korrekte Interpretation gewährleistet.

6.3 Interpretation der BASIC-Befehle

Wie nach Verlassen der Eingabewarteschleife ist auch bei der Interpretation der verschlüsselten BASIC-Befehle der Ablauf wieder davon abhängig, ob ein ganzes BASIC-Programm mit RUN bzw. GOTO XXX gestartet wird oder nur eine Direktkommandozeile ausgeführt werden soll.
In beiden Fällen jedoch holt die CHRGET-Routine (sie wurde bei der Initialisierung des Systems aus dem ROM- in den RAM-Bereich kopiert) die verschlüsselten BASIC-Befehle.
Bei einem Direktkommando wird dazu der CHRGET-Pointer (Programmzeiger) in (119,120) auf den Beginn des Eingabepuffers-1 (=511) gerichtet. Wird ein BASIC-Programm bearbeitet, so zeigt er auf den Anfang des Programmspeichers-1 (normalerweise 1024) bzw. auf den Anfang der BASIC-Zeile -1, die mit GOTO XXX bzw. RUN XXX adressiert wird.
Bei einem Direktkommando erfolgt sofort die Interpretation des ersten BASIC-Befehls, bei einer Programmzeile dagegen werden zuvor noch folgende Funktionen ausgeführt:

1. Es wird überprüft, ob die STOP-Taste gedrückt wurde. Ist dies der Fall, so werden alle Kanäle geschlossen, der Zeiger in den Tastaturpuffer zurückgesetzt und die BREAK-Meldung ausgegeben.
   Weiterhin wird noch die aktuelle Zeilennummer in (56,57) für "CONT" aufbewahrt. Danach erfolgt ein Sprung in den BASIC-Warmstart.

2. War die Stoptaste nicht gedrückt, so wird das High-Byte der Verkettungsadresse auf den Wert 0 abgefragt (siehe Speicherverwaltung 3.2.1).
   Hat es den Wert 0, so wird der Programmlauf beendet. Nach Ausgabe der READY-Meldung erfolgt Rückkehr zur Eingabewarteschleife.

3. Bei einer gültigen BASIC-Zeile wird die neue Zeilennummer nach (54,55) gebracht. Sie ist im Programmspeicher direkt hinter der Verkettungsadresse zu finden.
   Danach wird der Programmzeiger auf eine Stelle vor dem ersten BASIC-Befehl in der jeweiligen BASIC-Zeile gesetzt.

Bevor die Interpretation eines BASIC-Befehls ausgeführt wird, wird die CHRGET-Routine aufgerufen:
Diese inkrementiert den Programmzeiger und übernimmt das Byte in den Akku, auf das dieser dann zeigt. Nach dem Rücksprung befindet sich also im Akku entweder ein BASIC-Befehlskode bzw. bei einer Variablendefinition ohne LET ein ASCII-Zeichen.

In der Interpretationsphase wird nun vom Akkuinhalt der Betrag 128 abgezogen. Ergibt sich ein Wert kleiner als 0, so handelt es sich nicht um einen BASIC-Befehl (siehe Verschlüsselung), weshalb die Routine zur Variablen-definition aufgerufen wird.
Diese Routine überprüft zunächst, ob es sich wirklich um eine Variablendefinition handelt oder ob eine unsinnige ASCII-Zeichenfolge vorliegt. Im ersten Fall wird die Variable angelegt bzw. der angegebene Wert einer bereits bestehenden Variablen zugewiesen. Im zweiten Fall wird die Fehlermeldung "SYNTAX ERROR" ausgegeben mit darauffolgendem Sprung zur Eingabewarteschleife.

Wurde ein BASIC-Kode erkannt, so wird getestet, ob ein primärer BASIC-Befehl vorliegt. Wenn nicht, wird "SYNTAX ERROR" gemeldet.
Um bei einer Bejahung die richtige Routine zur Ausführung des Befehls aufrufen zu können, ist im ROM eine weitere Tabelle (49152-49221 bzw. <45056-45125) abgelegt, in der die Anfangsadressen (LB,HB) jedes einzelnen primären BASIC-Befehls stehen. Diese Tabelle ist in ihrer Reihenfolge identisch mit der Tabelle, die die zulässigen BASIC-Befehle als ASCII-Zeichenfolge beinhaltet. (Mit ihrer Hilfe wird in der Verschlüsselungsroutine ein BASIC-Befehl identifiziert. Dabei errechnete sich der entsprechende BASIC-Kode aus dem Wert des Wortzählers + 128).
Um nun also die Anfangsadresse der aufzurufenden BASIC-Routine zu erhalten, wird das Ergebnis der vorangegangenen Subtraktion (Akkuinhalt - 128) verdoppelt. Der Wert, der sich dabei ergibt, wird als Offsetzeiger in die Tabelle der Anfangsadressen der BASIC-Routinen verwendet. Er zeigt also auf das Low-Byte der Startadresse für die gewünschte Routine. Diese Adresse wird dann auf den Stapel (HB,LB) geschoben und durch RTS die jeweilige BASIC-Routine aufgerufen (siehe: Stack des CBM).
Die sekundären BASIC-Befehle (Funktionsanweisungen wie SIN, TAN, MID$, RIGHT$, LEFT$, PEEK, RND etc.) werden beim Auftreten innerhalb dieser Routinen auf ihre Zulässigkeit geprüft. Die Anfangsadressen der zugehörigen Verarbeitungsroutinen stehen genauso wie bei den primären BASIC-Befehlen in einer Tabelle (49222-49268 bzw. <45158-45203).
Der Aufruf jedoch erfolgt nicht vom Stapel her, sondern durch einen JMP-Befehl in der Zeropage (76). Die jeweilige Anfangsadresse (LB,HB) wird zuvor nach 77,78 transferiert.

Nach der Ausführung eines kompletten BASIC-Befehls wird zur Interpreterschleife zurückgekehrt, sofern in den einzelnen Verarbeitungsroutinen kein Fehler aufgetaucht ist.

In der Interpreterschleife wird die beschriebene Prozedur erneut durchlaufen, wobei das Vorhandensein des nötigen Trennzeichens bei mehreren BASIC-Befehlen je Programmzeile überprüft wird. Bei einer Folge von mehreren BASIC-Befehlen ohne Trennungszeichen wird "SYNTAX ERROR" gemeldet und in den READY-Modus verzweigt.

Dadurch, dass jeder primäre BASIC-Befehl in der Interpretationsschleife stets durch die CHRGET-Routine geholt wird und diese in der Zeropage steht (112-135), ist es relativ leicht möglich, eigene BASIC-Befehle zu schaffen. Dazu muss die CHRGET-Routine "angezapft" werden, d.h. bevor zur Interpretationsschleife zurückgesprungen wird, muss zuerst das entsprechende Maschinenprogramm durchlaufen werden, in dem eine Abfrage auf die neu entwickelten BASIC-Befehle stattfindet. Fällt dieser Test negativ aus, so wird die normale Interpretation ausgeführt, im anderen Fall wird nach der Verarbeitung des neuen BASIC-Befehls in die Interpreterschleife zurückgesprungen. Dabei ist als Einsprungsadresse 50941<46978 zu wählen. Vor diesem Rücksprung muss der Programmzeiger (119,120) noch auf den nächsten Doppelpunkt bzw. auf das Ende (0) im Inputpuffer gerichtet werden, da sonst "SYNTAX ERROR" erfolgt.
Diese Beschreibung stellt eine prinzipiell mögliche Anleitung zur "Kreation" neuer BASIC-Befehle dar. Selbstverständlich ist dabei ein genaues Selbststudium der CHRGET-Routine (112-135) und der Interpretationsschleife (50884-50989 bzw. <46922-47030) unbedingt Voraussetzung.
Auf das Einbinden neuer BASIC-Befehle wird im Kapitel "Erweiterung des Betriebssystems" noch näher eingegangen.

A. RUN (Kaltstart)

Unter Kaltstart eines BASIC-Programmes versteht man einen Programmstart mit gleichzeitiger Löschung aller Variablen.

Um einen solchen Start von einem Maschinenprogramm vornehmen zu können, muss folgende Befehlssequenz verwendet werden:

```
JSR  RUNC      (50546 < 46569)
JMP  RUNBUS    (50906 < 46943)
```

Ablauf der Routinen

Der Programmzeiger (CHRGET-Pointer in 119,120) wird auf den Wert des Programmanfangzeigers (40,41) minus 1 gesetzt (normalerweise 1024).
Danach werden alle Variablen gelöscht, indem die Variablenzeiger zurückgesetzt werden (siehe Speicherverwaltung).
Es erfolgt dann ein Sprung in die Interpreterschleife, in der das BASIC-Programm zur Ausführung gebracht wird (siehe BASIC-Interpretation).
Nach Beendigung des BASIC-Programmes erscheint die READY-Meldung. Das System befindet sich dann wieder in der Eingabewarteschleife.

B. RUN (Warmstart) = GOTO ANFANG

Unter Warmstart eines BASIC-Programmes versteht man einen Programmstart ohne vorheriges Löschen der Variablen.
Da die Variablen erhalten bleiben, kann also mit den Werten, die in einem vorhergehenden Programmlauf bestimmten Variablen zugewiesen wurden, weitergearbeitet werden.

Folgende Befehlssequenz ist dafür notwendig:

```
JSR STXPTP     (50599 < 46626)
JMP RUNBUS     (50906 < 46943)
```

Ablauf der Routinen

Der Ablauf unterscheidet sich von dem bei einem Kaltstart dadurch, dass vor dem Einsprung in die Interpreterschleife nur der Programmzeiger zurückgesetzt wird.

C. Abfrage der STOP-Taste

Es empfiehlt sich, in Assembler-Programmen Abfragen auf die STOP-Taste einzubauen, da diese Abfrage während der Ausführung eines Maschinenprogrammes standardmässig nicht durchgeführt wird.
Durch diese Abfrage ist es dann möglich, die Unterbrechung eines Assemblerprogrammes an bestimmten, gewünschten Stellen zuzulassen.

Dafür bietet das Betriebssystem zwei Möglichkeiten an:

a. ISCNT — beide Systeme: JSR $FFE1 (=65505)

Diese Routine testet die STOP-Taste.
Ist sie gedrückt, so erfolgt ein Sprung in den READY-Mode.
Ist sie nicht gedrückt, so wird in das aufrufende Programm zurückgekehrt.

b. ISCN1 — 62209 < 62261

Von dieser Routine wird mit dem Ergebnis des Tests im Statusregister an die aufrufende Routine zurückgekehrt:

STOP-Taste gedrückt : Z-Flag gesetzt
STOP-Taste nicht gedrückt : Z-Flag gelöscht

Zu beachten ist hierbei, dass bei einem positiven Test alle I/O-Kanäle geschlossen werden und die Tastatur bzw. der Bildschirm als Standard-Eingabe- bzw. Ausgabekanal festgesetzt werden.

Erweiterung des Betriebssystems

Je mehr man sich mit dem CBM befasst, desto häufiger entsteht der Wunsch, das Betriebssystem noch komfortabler zu gestalten.
Dies kann durch Maschinenprogramme erreicht werden. Allerdings sind die BASIC-Befehle für deren Aufruf nicht besonders komfortabel, da die Werte für die Einsprung-Adressen nicht leicht zu merken sind.
Dafür wären Befehle geeignet, deren Syntax sich leichter einprägen lässt als ein Zahlenwert.

Um den Befehlssatz des CBM zu erweitern, bieten sich 3 Ansatzpunkte an, die es erlauben, in das Betriebssystem des Rechners einzugreifen. Diese Punkte sind bestimmte Vektoren in der Zeropage, die auf Routinen des Betriebssystems zeigen.

Diese Vektoren können, da sie ja im RAM liegen, abgeändert werden, um auf eigene Routinen zu zeigen. Damit ist die Möglichkeit gegeben, beispielsweise neue BASIC-Befehle zu schaffen oder den Befehlssatz des Monitors zu erweitern.

Dazu bieten sich die folgenden Manipulationen an:

a. Verändern der CHARACTER-GET-ROUTINE (s.10):
   Diese Routine steht im Betriebssystem im Bereich (57593-57616 < 54169-54192) und wird bei der Initialisierung nach dem Einschalten in den Bereich (112-135=$70-87) kopiert.
   Sie holt das nächste Zeichen (TOKEN oder ASCII, siehe Kapitel.6) und erhöht den CHRGET-Zeiger ($77,78) um den Wert 1. Und das sowohl im DIREKT-Modus als auch im Programmablauf.
   Auf diese Möglichkeit der Implementierung wird im Beispiel A. näher eingegangen.

b. Verbiegen der Interruptroutine:
   Auch sie besitzt in der Zeropage einen Vektor, den sogenannten IRQ-Vektor, dessen Inhalt ebenfalls verstellt werden kann, da er im RAM liegt.
   Diese Möglichkeit der Erweiterung wird im Beispiel B. erläutert.

c. Die Monitor-Warteschleife:
   Die Warteschleife erwartet ein Zeichen und vergleicht es mit der Befehlstabelle, um danach die entsprechenden Routinen auszuführen. Bei Auffinden eines ungültigen Kommandos führt sie die ERROR-Routine aus (Ausgabe eines "?") und wartet auf das nächste Zeichen.
   Auch die Warteschleife wird durch einen Vektor in der Zeropage (1018,1019=$03FA,03FB) adressiert.
   Dieser Vektor zeigt normalerweise auf 59383=$E7F7 (< 58453=$E455). Durch Verändern dieses Wertes kann statt der Monitor-Routine ein beliebiges Programm adressiert werden.

Bei Änderung der Pointer muss das neue Programm selbstverständlich die Syntax für die neuen Befehle auch selbst überprüfen. Wird eine Zeichenfolge gefunden, die keinen der neuen Befehle darstellt, so muss in die normalen Routinen des Betriebssystems zurückgesprungen werden. Sonst ist ein ordnungsgemässer Ablauf der serienmässigen Kommandos nicht mehr gewährleistet !

Um zu zeigen, wie man die CHRGET-Routine "anbohrt", wird sie im Monitor betrachtet:

Die Speicherplätze $0078-007B zeigen einen Teil der Routine:

```
.M 0078,007F
.: 02 C9 3A B0 0A C9 20 F0
```

Diese Speicherplätze werden umgeändert in:

```
.: 02 4C 3D 03 0A 4C 41 03
```

Dabei wurden 2 Vergleiche der Routine einfach mit jeweils einem Sprung in das neue Maschinenprogramm überschrieben.
Die Veränderungen werden beim Maschinenprogramm durch die Routine INIT durchgeführt. Dies ist aber auch manüll im TIM möglich.

Das Maschinenprogramm muss nun die Vergleiche durchführen und kann dabei auch eigene Befehle berücksichtigen.
Um die CHRGET-Routine nicht unnötig zu verlangsamen ist allen neuen Befehlen ein (') zur Identifikation vorangestellt.
Wird diese Zeichen nicht gefunden, so wird die Abfrage der normalen Befehle angesprungen.
Bei Auffinden des '-Zeichens muss ein "neuer" Interpreter die Syntax des Befehles prüfen und die jeweiligen Routinen anspringen.

Mit dem beschriebenen Beispiel soll der Befehl 'CU ZN,SP eingeführt werden. Bei seiner Ausführung wird der Cursor auf Zeile ZN und Spalte SP gesetzt.
Dies ist sowohl im DIREKT-Modus als auch per Programm möglich.

Die Manipulation des Cursors ist aus dem Assemblerlisting ersichtlich.

```
                    0001 ;BEISPIEL A: VERAENDERN DER CHRGET-
                    0002              ;ROUTINE ZUM EINFUEGEN EIGENER
                    0003              ;BASIC-BEFEHLE.
                    0004              ;---------------------------------
                    0005 CHRGET       .DE $70 ;HOLT NAECHSTES ZEICHEN
                    0006 CHPOINT      .DE $79 ;POINTER FUER CHRGET-
                    0007                      ;ROUTINE
                    0008              .BA $033A
033A- A9 4C         0009 INIT         LDA #$4C          ;INITIALISIERUNG
033C- 85 79         0010              STA *CHPOINT
033E- 85 7D         0011              STA *CHPOINT+4
0340- A9 52         0012              LDA #L,IN1        ;CHRGET-POINTER
0342- 85 7A         0013              STA *CHPOINT+1    ;AUF NEUE BEFEHLE
0344- A9 03         0014              LDA #H,IN1        ;EINSTELLEN
0346- 85 7B         0015              STA *CHPOINT+2
0348- A9 56         0016              LDA #L,IN2
034A- 85 7E         0017              STA *CHPOINT+5
034C- A9 03         0018              LDA #H,IN2
034E- 85 7F         0019              STA *CHPOINT+6
0350- 60            0020              RTS
0351- 4C 70 00      0021 NEXT         JMP CHRGET    ;SPRUNG ZUR ROUTINE
0354- C9 3A         0022 IN1          CMP #$3A      ;1.EINSPRUNGSTELLE
0356- B0 0E         0023              BCS RETURN
0358- C9 20         0024 IN2          CMP #$20      ;2.EINSPRUNGSTELLE
035A- F0 F5         0025              BEQ NEXT
035C- C9 27         0026              CMP #$27      ;= (') ?
035E- F0 07         0027              BEQ INT       ;JA:=>INTERPRETER
0360- 38            0028              SEC
0361- E9 30         0029              SBC #$30
0363- 38            0030              SEC
0364- E9 D0         0031              SBC #$D0
0366- 60            0032 RETURN       RTS
0367- E0 00         0033 INT          CPX #0        ;I N T E R P R E T E R
0369- D0 09         0034              BNE CONT      ;FUER DIE NEUEN
                                                    ;BEFEHLE:
036B- C0 01         0035              CPY #1        ;UNTERSCHEIDET FOLG.
                                                    ;FAELLE
036D- D0 05         0036              BNE CONT      ;DIRECT MODE:=>
036F- C9 FF         0037              CMP #$FF      ;PROGRAMM-ABLAUF:=>
0371- D0 01         0038              BNE CONT      ;PROGRAMMEINGABE:=>INT
0373- 60            0039              RTS         ;ZURUECK ZUM INTERPRETER
0374- 20 70 00      0040 CONT         JSR CHRGET    ;HOLT NAECHSTES
                                                    ;ZEICHEN
0377- C9 43         0041              CMP #$43      ;='C' ?
0379- D0 41         0042              BNE FEHL1     ;NEIN:=>SYNTAX ERROR
037B- 20 70 00      0043              JSR CHRGET    ;NAECHSTES ZEICHEN
037E- C9 55         0044              CMP #$55      ;='U' ?
0380- D0 3A         0045              BNE FEHL1     ;NEIN:=>SYNTAX ERROR
0382- 20 C4 03      0046              JSR INPUT     ;NAECHSTE ZAHL
0385- C9 2C         0047              CMP #$2C      ;=KOMMA ?
0387- D0 33         0048              BNE FEHL1     ;NEIN:=>SYNTAX ERROR
0389- AC FF 00      0049              LDY $FF       ;LADEN DER
                                                    ;EINGELESENEN
038C- 38            0050              SEC           ;Zahl
038D- C0 19         0051              CPY #25       ;ZEILENZAHL >24 ?
038F- B0 30         0052              BCS FEHL2     ;JA:=>ILLEGAL
                                                        QUANTITIY
```

```
0391- 20 C4 03  0053              JSR INPUT     ;NAECHSTE ZAHL
0394- AD FF 00  0054              LDA $FF       ;LADEN DER ZAHL
0397- 38        0055              SEC
0398- C0 28     0056              CPY #40       ;SPALTENZAHL >39 ?
039A- B0 25     0057              BCS FEHL2     ;JA:=>ILLEGAL
                                                ;      QUANTITY
039C- 8D C6 00  0058              STA 198
039F- A9 00     0059              LDA #0
03A1- A2 80     0060              LDX #128
03A3- 8C D8 00  0061              STY 216
03A6- C0 00     0062              CPY #0
03A8- F0 09     0063              BEQ AUS       ;C U R S O R -
03AA- 18        0064 ADD          CLC
03AB- 69 28     0065              ADC #40
03AD- 90 01     0066              BCC NIC  ;M A N I P U L A T I O N
03AF- E8        0067              INX
03B0- 88        0068 NIC          DEY
03B1- D0 F7     0069              BNE ADD
03B3- 8D C4 00  0070 AUS          STA 196
03B6- 8E C5 00  0071              STX 197
03B9- 4C 76 00  0072              JMP $76
03BC- A2 10     0073 FEHL1        LDX #$10      ;FEHLERMELDUNG:
03BE- 4C 57 C3  0074              JMP 50007 <46031 ;SYNTAX ERROR
03C1- A2 35     0075 FEHL2        LDX #$35      ;FEHLERMELDUNG:
03C3- 4C 57 C3  0076              JMP 50007 <46031 ;ILLEGAL QUANTITY
03C6- 20 70 00  0077 INPUT        JSR CHRGET    ;NAECHSTES ZEICHEN
03C9- 38        0078              SEC           ;LESEN
03CA- E9 30     0079              SBC #$30      ;UMRECHNUNG IN
03CC- AA        0080              TAX           ;SPALTENNUMMER
03CD- 20 70 00  0081              JSR CHRGET    ;NAECHSTES ZEICHEN
03D0- 38        0082              SEC           ;LESEN
03D1- E9 30     0083              SBC #$30      ;UMRECHNUNG IN
03D3- E0 00     0084              CPX #0        ;ZEILENNUMMER
03D5- F0 04     0085              BEQ NAD       ;BERECHNUNG DER HEX-
03D7- 18        0086              CLC           ;DARSTELLUNG
03D8- 69 0A     '0087 ADZ         ADC #10       ;DER ZWEISTELLIGEN
03DA- CA        0088              DEX           ;DEZIMALZAHL
03DB- 8D FF 00  0089 NAD          STA $FF       ;ABSPEICHERN IN $FF
03DE- 20 70 00  0090              JSR CHRGET    ;NAECHSTES ZEICHEN
03E1- 60        0091              RTS
                0092              .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/CHRGET=0070          /CHPOINT=0079          INIT=033A
NEXT=0351             IN1=0354               IN2=0358
RETURN=0366           INT=0367               CONT=0374
ADD=03AA              NIC=03B0               AUS=03B3
FEHL1=03BC            FEHL2=03C1             INPUT=03C6
ADZ=03D8              NAD=03DB
```

Das Beispiel B. beschreibt, wie man durch Verbiegen des IRQ-Vektors das Einfügen einer nützlichen Routine in das Betriebssystem des CBM erreichen kann:

Sobald die Taste 'Pfeil nach links' gedrückt wird, wird jedes laufende Programm (BASIC oder Assembler) angehalten und eine Hardcopy des momentanen Bildschirminhaltes ausgegeben auf den Drucker am IEC-Bus.
Dabei wird der Bildschirm originalgetreu wiedergegeben, d.h. es werden 24 Zeilen mit je 40 bzw. 80 Zeichen ausgegeben.

Diese Routine ist bestens dazu geeignet, beispielsweise ein Programm zu dokumentieren.

Um diese Funktion zu erreichen, wird MIT SYS 826 der IRQ-Vektor ($90,91) auf die eigene Routine gestellt. Dies hat zur Folge, dass alle 1/60 Sekunden anstatt der Tastaturabfrage und Uhrsteuerung in diese Routine eingesprungen wird.

Dabei wird nun im Beispielprogramm abgefragt, ob die Taste 'Pfeil links' gedrückt wird.
Fällt der Test negativ aus (Taste nicht gedrückt), so wird bei der normalen IRQ-Routine fortgesetzt (=Sprung nach $E62E<E455).

Bei betätigter Taste wird zunächst das eingefügte Programm abgearbeitet (in diesem Falle: eine Hardcopy ausgegeben) und dann erst in die IRQ-Routine gesprungen.

Der Rechner befindet sich dann im gleichen Zustand wie vor der Unterbrechung durch die Hardcopy.

```
                    0001            ;BEISPIEL B.
                    0002            ;-----------
                    0003 IRQVEKT    .DE $90
                    0004 IRQNORM    .DE $E62E <E455 ;NORMALER IRQ-VEKT.
                    0005 BILD       .DE $8000       ;ANFANG DES BILD-
                    0006                            ;SCHIRMSPEICHERS
                    0007 ZWISP      .DE $1F         ;ZWISCHENSPEICHER
                    0008 POINT      .DE $21         ;ZEIGER AUF AKTUELLES
                    0009                            ;BILDSCHIRMZEICHEN
                    0010 AUSDRU     .DE $FFD2       ;AUSGABE AUF DEN
                                                    ;DRUCKER
                    0011            ;--------------------------------
                    0012            .BA $033A
033A- 78            0013 INIT       SEI             ;INITIALISIERUNG
033B- A9 45         0014            LDA #L,HARD     ;---------------
033D- 85 90         0015            STA *IRQVEKT    ;IRQ-VEKTOR AUF DAS
033F- A9 03         0016            LDA #H,HARD     ;HARDCOPY-PROGRAMM
0341- 85 91         0017            STA *IRQVEKT+1      ;EINSTELLEN
0343- 58            0018            CLI
0344- 60            0019            RTS
                    0020            ; H  A  R  D  C  O  P  Y
0345- A5 97         0021 HARD       LDA *$97    ;GEDRUECKTES ZEICHEN
0347- C9 4B         0022            CMP #$4B    ;= PFEIL NACH LINKS ?
0349- D0 03         0023            BNE NORMAL  ;NEIN:=> NORMAL WEITER
034B- 20 51 03      0024            JSR COPY    ;JA:=> HARDCOPY
034E- 4C 2E E6      0025 NORMAL     JMP IRQNORM
0351- A9 00         0026 COPY       LDA #L,BILD ;ANFANG DES
0353- 85 1F         0027            STA *ZWISP  ;BILDSCHIRM-RAMS
0355- A9 80         0028            LDA #H,BILD ;ZWISCHENSPEICHERN
0357- 85 20         0029            STA *ZWISP+1
0359- A9 04         0030            LDA #$04    ;NR. 4:
035B- 85 B0         0031            STA *$B0    ;=AUSGABEGERAET
                                                ;  (=DRUCKER)
035D- 85 D4         0032            STA *$D4    ;=FILE-NUMMER
035F- 20 BA F0      0033            JSR $F0BA    ;OPEN IEC-BUS
0362- 20 2D F1      0034            JSR $F12D    ;TEST IEC-BUS
0365- A9 19         0035            LDA #$19
0367- 85 22         0036            STA *POINT+1 ;ZEIGER SETZEN
0369- A9 0D         0037 CONT       LDA #$0D
036B- 85 21         0038            STA *POINT
036D- 20 D2 FF      0039            JSR AUSDRU
0370- A9 11         0040            LDA #17
0372- AE 4C E8      0041            LDX $E84C    ;BILDSCHIRM IM
0375- E0 0C         0042            CPX #12      ;GRAPHIKMODUS ?
0377- D0 02         0043            BNE MARKE1   ;JA:=>
0379- A9 90         0044            LDA #144
037B- 20 D2 FF      0045 MARKE1     JSR AUSDRU   ;CHR$(17) AN DRUCKER
037E- A0 00         0046            LDY #$00
0380- B1 1F         0047 MARKE2     LDA (ZWISP),Y ;ZEICHEN VOM
BILDSCHIRM
0382- 29 7F         0048            AND #%01111111 ;BIT 7 AUSBLENDEN
0384- AA            0049            TAX
0385- B1 1F         0050            LDA (ZWISP),Y
0387- 45 21         0051            EOR *POINT
0389- 10 0B         0052            BPL MARKE3
038B- B1 1F         0053            LDA (ZWISP),Y
038D- 85 21         0054            STA *POINT
038F- 29 80         0055            AND #%10000000 ;SENDET AN DEN
0391- 49 92         0056            EOR #%10010010 ;DRUCKER:
                    0057            ;BEI GROSSBUCHSTABEN: CHR$(19)
                    0058            ;BEI KLEINBUCHSTABEN: CHR$(147)
0393- 20 D2 FF      0059            JSR AUSDRU
```

```
0396- 8A        0060 MARKE3     TXA
0397- C9 20     0061            CMP #$20      ;DRUCKBARES ZEICHEN ?
0399- B0 04     0062            BCS MARKE4    ;JA:=>
039B- 09 40     0063            ORA #%01000000
039D- D0 0E     0064            BNE MARKE6
039F- C9 40     0065 MARKE4     CMP #$40      ;GROSSBUCHSTABE ?
03A1- 90 0A     0066            BCC MARKE6    ;NEIN:=>
03A3- C9 60     0067            CMP #$60      ;KLEINBUCHSTABE ?
03A5- B0 04     0068            BCS MARKE5    ;NEIN:=>
03A7- 09 80     0069            ORA #%10000000
03A9- D0 02     0070            BNE MARKE6
03AB- 49 C0     0071 MARKE5     EOR #%11000000
03AD- 20 D2 FF  0072 MARKE6     JSR AUSDRU
03B0- C8        0073            INY
03B1- C0 28     0074            CPY #$28 <50 ;ZEILENENDE ERREICHT ?
03B3- 90 CB     0075            BCC MARKE2    ;NEIN:=>
03B5- A5 1F     0076            LDA *ZWISP    ;JA:=> ZEIGER AUF
03B7- 69 27     0077            ADC #39       ;BILDSCHIRMZEILE
03B9- 85 1F     0078            STA *ZWISP    ;ERHOEHEN
03BB- 90 02     0079            BCC MARKE7
03BD- E6 20     0080            INC *ZWISP+1
03BF- C6 22     0081 MARKE7     DEC *POINT+1
03C1- D0 A6     0082            BNE CONT
03C3- A9 0D     0083            LDA #$0D      ;CARRIAGE RETURN
03C5- 20 D2 FF  0084            JSR AUSDRU
03C8- 4C CC FF  0085            JMP $FFCC     ;RESET I/O
                0086            .EN
```

```
LABEL FILE:   ( / = EXTERNAL )

/IRQVEKT=0090          /IRQNORM=E62E <E455     /BILD=8000
/ZWISP=001F            /POINT=0021             /AUSDRU=FFD2
INIT=033A              HARD=0345               NORMAL=034E
COPY=0351              CONT=0369               MARKE1=037B
MARKE2=0380            MARKE3=0396             MARKE4=039F
MARKE5=03AB            MARKE6=03AD             MARKE7=03BF
```

	62819 $F563	62756 $F524	E:($D1)=Länge des Filenamens ($D2)=Laufende Filenummer ($D3)=Sekundär-adresse ($D4)=Gerätenummer ($DA)=LB Anfang d. ($DB)=HB Filenamens	AC XR YR	Eröffnet Datei mit den angegebenen Parametern. Grundvoraussetzung für fast alle weiteren Operationen mit dem IEC-Bus ! (OPEN)
	62176 $F2E0	62124 $F2AC	E:($D2)=Nummer der zu schliessenden Datei.	AC XR	Schliesst die angegebene Datei. (CLOSE)
JLTLK	62145 $F2C1	62093 $F28D	E:(AC)=Logische Filenummer A:(XR)=Index		Uebergibt den Index für die Stelle des Files in der Filetabelle im XR.
JZ100	62157 $F2CD	62105 $F299	E:(XR)=Index A:($D2) Parameter ($D3) des laufen- ($D4) den Files.	AC	Setzt das File, dessen Index im XR steht, als laufendes File.
TALK	61650 $F0D2	61622 $F0B6	E:($D2)=Laufende Filenummer	AC	Das Gerät, das von der lfd. Filenummer angesprochen wird, wird zum TALKER.
LISTN	61653 $F0D5	61626 $F0BA	E:($D2)=Laufende Filenummer	AC	Das Gerät, das von der lfd. Filenummer angesprochen wird, wird zum LISTENER.
UNTLK	61870 $F1AE	61823 $F17F		AC	Setzt alle TALKER auf ihre Grundstellung. (UNTALK)
UNLSN	61881 F1B9	61827 $F183		AC	Setzt alle LISTENER auf ihre Grundstellung. (UNLISTEN)
CLRCH	62118 F2A6	62066 $F272		AC	Setzt Ein/Ausgabe auf den Normalzustand: Eingabe von Tastatur, Ausgabe auf Bildschirm.

Ausgabe auf den IEC-Bus:

Voraussetzung

Das Ausgabegerät muss über das lfd. File anzusprechen sein ! (OPEN IEC-Bus)

CKOUT	63486 F7FE	63420 $F7BC	E:(XR)=Logische Filenummer		Lenkt alle Ausgaben für den Bildschirm um auf das angesprochene Ausgabegerät. (CMD)
SECND	61763 $F143	61736 $F128	E:(AC)=Sekundär-adresse	AC	Sendet den Wert im AC im Kommandomodus als Sekundäradresse an das aktuelle File.
CIOUT	61854 $F19E	61807 $F16F	E:(AC)=Zu sendendes Zeichen ($D3)=Log.Filenr. ($D4)=Sekundäradr. ($D5)=Gerätenummer		Sendet Zeichen im AC an das laufende Ausgabegerät.

Ausgabe auf den Bildschirm A:

Voraussetzung

Der Bildschirm muss das aktuelle Ausgabegerät sein ! =Zustand nach Einschalten und nach Ausführen von JSR CLRCH.

BSOUT	62054 $F266	62002 $F232	E:(AC) =Zeichen		Gibt ein (ASCII-) Zeichen im AC ab der aktuellen Cursor-Position aus.
STROUT	47901 $BB1D	51740 $CA1C	E:(AC)=LB(Anfangs- (YR)=HB adresse)	AC XR YR	Gibt den ASCII-String ab der Anfangsadresse aus. Der String muss mit dem Byte $00 enden!
	47908 $BB24	51747 $CA23	E:($1F)=LB(Anfangs- ($20)=HB adresse) (XR) =Zeichenzahl	AC XR	Gibt den ASCII-String, dessen Zeichenzahl im XR steht, auf den Bildschirm aus.
ERROR	46031 $B3CF	50007 $C357	E:(XR)=Offsetadresse (Keine Subroutine!)	AC XR YR	Gibt eine Fehlermeldung aus. Siehe Anhang 6
MSG	61829 $F185	61782 $F156	E:(YR)=Offsetadresse (Subroutine)	AC YR	Gibt eine Fehlermeldung aus dem Bereich $F000-F0B5 aus (Siehe Anhang 6)
WRTWO	55089 $D731	59268 $E784	E:(XR)=1.Zeichen (AC)=2.Zeichen	YR	Gibt 2 Zeichen, die in XR und AC stehen, auf den Bildschirm aus.
PRT	57858 $E202	58328 $E3D8	E:(AC)=Zeichen		Gibt ein ASCII-Zeichen im AC an Bildschirm aus mit Berücksichtigung der etwaigen Steuerfunktion.
		59114 $E6EA	E:(AC)=Zeichen		Gibt ein Zeichen im BSC-Code (siehe Anhang 7) ohne Berücksichtigung der etwaigen Steuerfunktion aus.
WROB	55074 $D722	59253 $E775	E:(AC)=Zeichen	XR	Gibt ein Zeichen im AC auf den Bildschirm aus als 2-stellige Hexadezimalzahl.
LINPRT	53123 $CF83	56537 $DCD9	E:(XR)=LB (AC)=HB	AC XR YR	Gibt eine ganze 2-Byte-Zahl zwischen 0-65536 in XR und AC dezimal auf den Bildschirm aus.

Ausgabe auf den Bildschirm B:

Voraussetzung

Der Bildschirm muss das aktuelle Ausgabegerät sein ! =Zustand nach Einschalten und nach Ausführen von JSR CLCHR.

WROA	55063 $D117	59242 $E76A	E: ($FB)=LB ($FC)=HB	XR	Gibt eine 2-Byte-Zahl als 4-stellige Hexadezimalzahl aus.
DM	54519 $D4F7	64915 $FD93	E: ($FB)=LB (Anfangs- ($FC)=HB adresse) (AC) =Anzahl	YR	Gibt die Inhalte von Speicherstellen ab der Anfangsadresse durch je ein Space getrennt hexadezimal aus. Die Anzahl der auszugebenden Stellen steht im AC.
		57943 $E257		XR AC	Stellt den Cursor auf die Ecke oben links des Bildschirms (=CURSOR HOME)
		58649 $E519		XR	Gibt eine Leerzeile nach der momentanen Cursorzeile aus. Wenn nötig (=letzte Zeile), Bildschirminhalt nach oben schieben. RVS-, Cursorposition und Zeilenlängeflags werden dabei nicht verändert.
		58268 $E39C		AC XR YR	Schiebt Bildschirm nach oben und setzt Cursor auf den Anfang der Zeile hinter der vorherigen Cursorzeile.
		58687 $E53F		AC XR YR	Schiebt Bildschirminhalt nach oben und setzt Cursor auf den Anfang der untersten Zeile.
		58275 $E3A3		AC XR	Schiebt Bildschirminhalt nach oben, wenn der Cursor vorher auf der untersten Zeile war. Ansonsten Cursor auf nächsten Zeilenanfang.

Voraussetzung

Der Bildschirm muss das aktuelle Ausgabegerät sein ! =Zustand nach Einschalten und nach Ausführen von JSR CLCHR.

	57365 $E015	57897 $E229		AC XR YR	Löscht den gesamten Bildschirminhalt. Cursor auf Ecke oben links.
		56547. $DCE3	E:($5E..63)	YR	Gibt den Inhalt des FAC#1 ab der aktuellen Cursor-position als Gleitkommazahl aus.
	64976 $FDD1	64973 $FDCD		AC	Gibt 1 Space aus.
SPAC2	54574 D52E	64970 $FDCA		AC	Gibt 2 Spaces aus.
		51776 $CA40		AC	Versetzt Cursor um eine Stelle nach rechts.
CRDO	47839 $BADF	51682 $C9E2		AC	Gibt CARRIAGE RETURN aus.
	47848 $BAE8	51687 $C9E5		AC	Gibt LINEFEED aus.
	48367 $BCEF	52212 $CBF4		AC XR YR	Gibt '?EXTRA IGNORED' aus.

Zusätzliche Bildschirmroutinen im BASIC 4.0

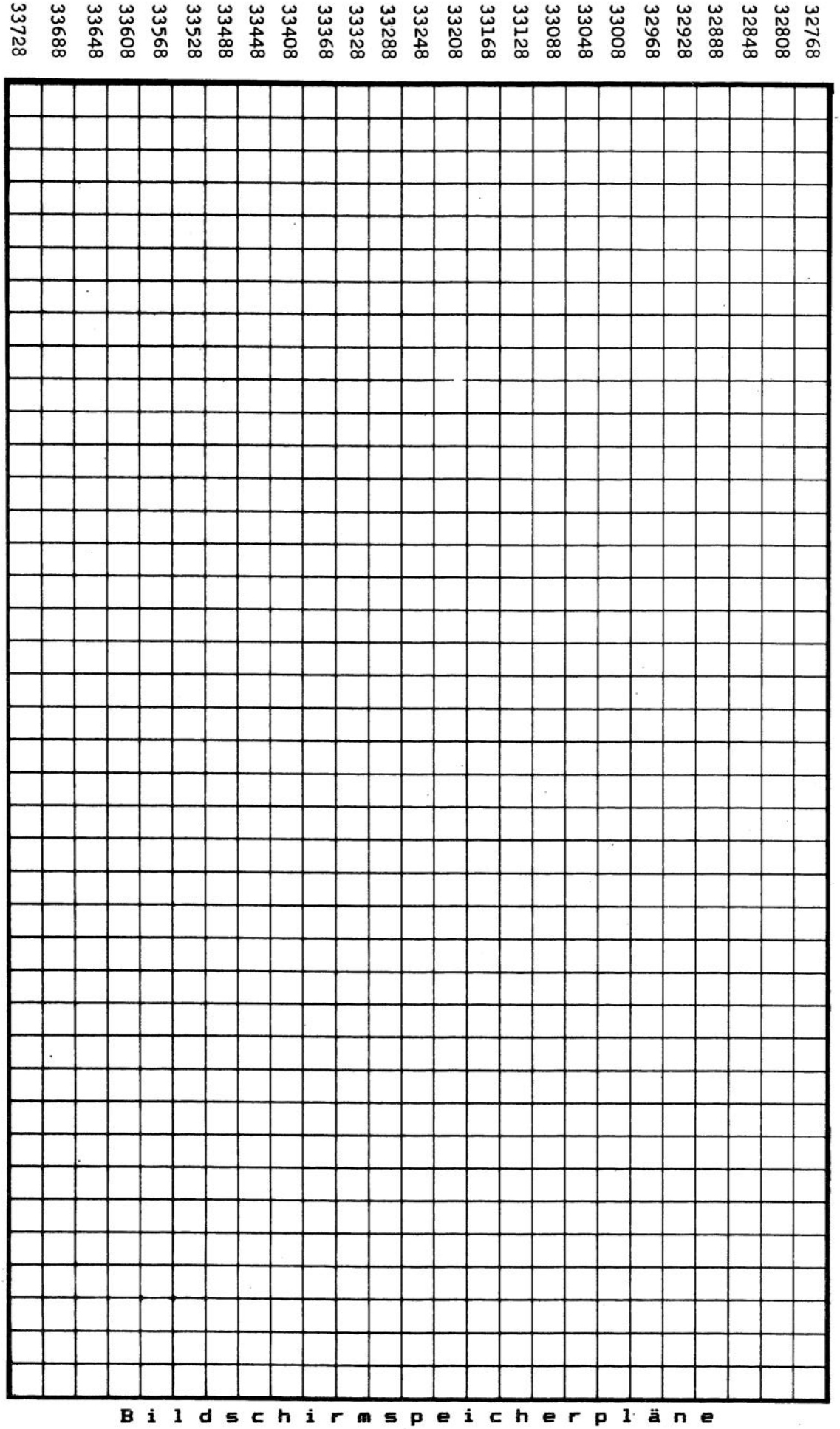
58344 $E3EB		AC XR YR	Rollt das vorher definierte Bildschirmfenster nach oben.
58312 $E3C8		AC XR YR	Rollt das vorher definierte Bildschirmfenster nach unten.
57793 $E1C1	E:($C4)= Zeilen-nummer (YR)= letzte verbleibende Stelle	AC YR	Löscht die definierte Zeile ab der Position (YR+1).
58262 $E7A0	E:($C4)= Zeilen-nummer ($C6)= Position d. letzten zu löschenden Zeichens	AC XR	Löscht die definierte Zeile bis zur angegebenen Position.
58087 $E2F3	E:($D8)= Zeilen-nummer	AC XR YR	Löscht die definierte Zeile und rollt den Rest des Bildschirmfensters nach oben.
58816 $E5C0	E:($D8)= Zeilen-nummer	AC XR YR	Macht die definierte Zeile frei und schiebt den Rest des Bildschirmfensters nach unten.
57466 $E07A		AC XR YR	Schaltet den Bildschirm um auf Textmodus. (Zeilenabstand 3 Bildpunkte)
57464 $E07B		AC XR YR	Schaltet den Bildschirm um auf Grafikmodus. (ohne Zeilenabstand)
59047 $E6C1		AC XR YR	Schlägt die eingebaute Glocke an.

Meldungen:

Adress-Offset	Meldung:
XR=$6B	BAD SUBSCRIPT
XR=$DB	CAN'T CONTINUE
YR=$74	DEVICE NOT PRESENT
XR=$85	DIVISION BY ZERO
XR=$F9	ERROR
XR=$BF	FILE DATA
YR=$24	FILE NOT FOUND
YR=$17	FILE NOT OPEN
YR=$0E	FILE OPEN
YR=$3D	FOR
XR=$C8	FORMULA TOO COMPLEX
YR=$A3	FOUND
XR=$95	ILLEGAL DIRECT
XR=$35	ILLEGAL QUANTITY
YR=$5F	LOAD
XR=$00	NEXT WITHOUT FOR
YR=$86	NOT INPUT FILE
YR=$94	NOT OUTPUT FILE
YR=$AA	OK
YR=$56	ON TAPE #
XR=$4D	OUT OF MEMORY
XR=$45	OVERFLOW
YR=$41	PRESS PLAY
YR=$4D	& RECORD
XR=$78	REDIM'D ARRAY
XR=$16	RETURN WITHOUT GOSUB
YR=$32	SEARCHING
XR=$B0	STRING TOO LONG
XR=$10	SYNTAX
YR=$00	TOO MANY FILES
XR=$A3	TYPE MISMATCH
XR=$E9	UNDEF'D FUNCTION
XR=$5A	UNDEF'D STATEMENT
YR=$6D	VERIFY
YR=$64	WRITING

Bildschirmcodes

@	0	9	57	┬	114	▒	170	▀	226
A	1	:	58	┤	115	▚	171	▄	227
B	2	;	59	├	116	▌	172	▐	228
C	3	<	60	┴	117	▖	173	▝	229
D	4	=	61	┼	118	▗	174	▘	230
E	5	>	62	▔	119	▞	175	▙	231
F	6	?	63	▁	120	▝	176	▟	232
G	7	─	64	▏	121	▘	177	▜	233
H	8	♠	65	▒	122	▙	178	▛	234
I	9	│	66	▕	123	▟	179	▞	235
J	10	─	67	▗	124	▜	180	▚	236
K	11	─	68	▘	125	▛	181	▌	237
L	12	─	69	▝	126	▞	182	▐	238
M	13	─	70	▚	127	▚	183	▖	239
N	14	│	71	▞	128	▛	184	▗	240
O	15	│	72	▜	129	▜	185	▘	241
P	16	╮	73	▟	130	▙	186	▝	242
Q	17	╲	74	▙	131	▟	187	▘	243
R	18	╯	75	▛	132	▚	188	▙	244
S	19	└	76	▜	133	▞	189	▟	245
T	20	╱	77	▞	134	▛	190	▜	246
U	21	╱	78	▚	135	▜	191	▛	247
V	22	┌	79	▌	136	▙	192	▞	248
W	23	┐	80	▐	137	▟	192	▚	249
X	24	●	81	▖	138	▛	193	▌	250
Y	25	─	82	▗	139	▜	194	▐	251
Z	26	♥	83	▘	140	▙	195	▖	252
[	27	│	84	▝	141	▟	196	▗	253
\	28	╭	85	▘	142	▙	197	▘	254
]	29	╳	86	▝	143	▟	198	▝	255
↑	30	○	87	▚	144	▛	199		
←	31	♣	88	▞	145	▜	200		
	32	│	89	▚	146	▛	201		
!	33	♦	90	▞	147	▜	202		
"	34	┼	91	▙	148	▟	203		
#	35	▒	92	▟	149	▙	204		
$	36	│	93	▛	150	▜	205		
%	37	π	94	▜	151	▛	206		
&	38	◥	95	▞	152	▚	207		
'	39		96	▚	153	▌	208		
(	40	▌	97	▞	154	▐	209		
)	41	▀	98	▛	155	▖	210		
*	42	▔	99	▜	156	▗	211		
+	43	_	100	▙	157	▘	212		
,	44	│	101	▟	158	▝	213		
-	45	▒	102	▛	159	▘	214		
.	46	│	103	▜	160	▝	215		
/	47	▒	104	▙	160	▘	216		
0	48	◤	105	▟	161	▝	217		
1	49	│	106	▛	162	▘	218		
2	50	├	107	▜	163	▝	219		
3	51	▪	108	▞	164	▘	220		
4	52	└	109	▚	165	▝	221		
5	53	┐	110	▌	166	▘	222		
6	54	▔	111	▐	167	▝	223		
7	55	┌	112	▖	168	▘	224		
8	56	┴	113	▗	169	▝	225		


33728
33688
33648
33608
33568
33528
33488
33448
33408
33368
33328
33288
33248
33208
33168
33128
33088
33048
33008
32968
32928
32888
32848
32808
32768
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
Bildschirmspeicherpläne

Eingabe vom IEC-Bus:

Voraussetzung

Das File, das durch diese Routine angesprochen werden soll, muss vorher geoeffnet und das von ihm adressierte Gerät als TALKER benannt worden sein !

ACPTR	61888 61836 $F1C0 $F18C	E:($D2)=Logische Filenummer ($D3)=Sekundär-adresse ($D4)=Gerätenummer A:(AC)=Zeichen vom IEC-Bus	AC	Holt ein Zeichen vom eingestellten IEC-Bus-Gerät und übergibt es im AC.
	62465 62528 $F401 $F440	E: ($96)=STATUS=0 ($D4)=Gerätenummer ($DA)=LB Anfangsadr ($DB)=HB Filename ($D1)=Länge d. " A: ($28)=LB Programm - ($29)=HB anfangsad. ($2A)=LB Programm - ($2B)=HB endadresse	AC XR YR	Lädt ein Programmfile von der Diskettenstation am IEC-Bus. Dieses kann dann mit RUN oder entspr. Subroutinen gestartet werden. Die BASIC - Pointer werden auf das Programm eingestellt.
	62348 62290 $F39C $F352	E: ($96)=STATUS=0 ($9D)=LOAD-Flag=0	AC XR YR	Lädt ein Programmfile von der Diskettenstation am IEC-Bus, ohne die BASIC-Pointer zu verändern. Es muß JSR TWAIT (=Abschluß) folgen !

Tastatur / Bildschirm-Eingabe:

Vorausetzung

Der IEC-Bus muss eingestellt sein auf
Ein/Ausgabe von Tastatur/Bildschirm !
= Normalzustand nach Einschalten oder
nach Aufruf der Routine CLCHR.

GETIN	61957 $F205	61905 $F1D1	A:(AC)=Zeichen von der Tastatur	AC	Holt erstes Zeichen aus dem Tastaturpuffer und rückt den übrigen Inhalt nach.
BASIN	61973 $F215	61921 $F1E1	E:($1F)=LB String- ($20)=HB adresse A:(AC)=Zeichen von Tastatur	AC	Erwartet die Eingabe einer Zeichenfolge, die mit RETURN endet, vom Bildschirm und legt sie ab der Stringadresse ab.
RDOB	55139 $D763	59318 $E7B7	A:(AC)=Hex-Byte Carry=1,wenn Space eingegeben.	AC	Holt zwei ASCII-Zeichen vom Bildschirm, und legt sie als 1 Hex-Byte im AC ab.
RDOA	55124 $D754	59303 $E7A7	A:($FB)=LB Adresse ($FC)=HB	AC XR	Holt eine 2-Byte-Hex- adresse vom Bildschirm und legt sie in die angegebenen Speicherzellen.
GETNUM	51489 $C921	54982 $D6C6	A:($11)=HB ($12)=LB (XR) =Wert	AC XR YR	Holt eine Adresse (0-65535), gefolgt von einem Wert (0-255), vom Bildschirm und über- prüft, ob Klammern und Komma (als Trennungszeichen) richtig gesetzt sind.
TIMNUM	47531 $B9AB	51496 $C928	FAC#1=($5E-63) A:(FAC#1)=FAC#1+Zahl vom Bild- schirm.	AC XR YR	Holt eine reelle Zahl vom Bildschirm und addiert sie zum Inhalt des FAC#1.
FRMEV1	48542 $BD9E	52389 $CCA4	FAC#1=($5E-63)	AC XR YR	Holt eine Fliesskommazahl vom Bildschirm und legt sie im Registerformat in den FAC#1 ab.

Ein/Ausgabe Kassettenrekorder:

FAH	62949 $F5E5	62886 $F5A6	E:($D4)=Rekorder-Nr. A:(Kassettenpuffer)= Fileheader	AC XR YR	Sucht Fileheader auf dem eingestellten Kassettenrekorder.
	63642 $F89A	63573 $F855	E:($D4)=Rekorder-Nr. A:(Kassettenpuffer)= Block	AC XR YR	Liest Block mit 192 Bytes vom eingestellten Rekorder in den entsprechenden TAPE-Puffer.
	63651 $F867	63582 $F85E	E:($D4)=Rekorder-Nr. ($FB)=LB Anfangs- ($FC)=HB adresse ($C9)=LB End- ($CA)=HB adresse+1	AC XR YR	Liest einen Block beliebiger Länge in den Speicherbereich zwischen Anfangs- und Endadresse.
CSTE1	63701 $FCBD	63632 $F890	E:($D4)=Rekorder-Nr. ($FB)=LB Anfangs- ($FC)=HB adresse ($C9)=LB End- ($CA)=HB adresse+1 ($0279)=$69=Prüf- summe	AC XR YR	Schreibt einen Block beliebiger Länge (zwischen Anfangs- und Endadresse) auf Band.
	62474 $F408	62409 $F3C9	E:($D4)=Rekorder-Nr. ($D1)=Länge des Filenamens ($DA)=LB Adresse des ($DB)=HB Filenamens		Lädt ein Programm vom eingestellten Rekorder.
SV5	63203 $F6E3	63140 $F6A4	E:($D4)=Rekorder-Nr. ($D1)=Länge des ($DA)=LB Adresse des ($DB)=HB Filenamens		Schreibt ein Programm auf den eingestellten Kassettenrekorder.
	63787 $D235	63718 $F8E6		AC	Beendet Ein/Ausgabe von/auf Band.

Ueberprüfung von Tasten:

CSTE1	63575 $F857	63506 $F812	E:($D4)=Rekorder-Nr.	AC XR YR	Ueberprüft Rekorder-tasten für Lesevorgang. (PRESS PLAY...).
	63628 $F88C	63559 $F847	E:($D4)=Rekorder-Nr.	AC XR YR	Ueberprüft Rekorder-tasten für Schreibvorgang. (PRESS PLAY & RECORD...)
CS10	63610 $F87A	63541 $F835	E:($D4)=Rekorder-Nr. A: Zero-Flag=1, wenn eine Taste gedrückt ist.	AC XR YR	Ueberprüft Rekorder-tasten ohne Textausgabe.
STOP1	62261 $F335	62209 $F301	A:Z-Flag gesetzt, wenn STOP gedrückt	AC	Ueberprüft, ob STOP-Taste gedrückt ist.
ISCNTC	65505 $FFE1	65505 $FFE1			Springt in READY-Mode, wenn STOP gedrückt.

56020 $DAD4	E: ($DA)=LB Anfangs- ($DB)=HB adresse ($D1)= Länge	AC XR YR	Baut den String ab der Anfangsadresse und der angegebenen Adresse als 1.Dateinamen in den Befehlsstring ein.
56033 $DAE1	E: ($FD)=LB Anfangs- ($FE)=HB adresse ($D1)= Länge	AC XR YR	Baut den String ab der Anfangsadresse und der angegebenen Adresse als 2.Dateinamen in den Befehlsstring ein.
56061 $DCFD	E: ($033F)=1.Zeichen ($0340)=2.Zeichen	AC XR	Baut die angegebenen Zeichen als Disk-ID in den Befehlsstring ein.
55437 $D88D	E: ($B0)=akt.Ausgabe- gerät ($0353..) Befehls- string	AC XR YR	Führt den Befehlsstring ab $0353 aus. (z.B.: Wiederholung der letzten Disk-Operation)
55857 $DA31	E: ($D2)=Filenummer ($11)=LB Record- ($12)=HB zeiger ($033A)=Stelle des Zeichens im RECORD.	AC XR YR	Stellt den RECORD-Zeiger auf das definierte Zeichen ein.
55687 $D97D	E: ($D4)= Geräte- nummer	AC XR YR	Legt den Status des angegebenen IEC-Bus-gerätes in ds$ ab.

FAC#1 = ($5E-63)
FAC#2 = ($66-6B)

CONUPK	52162 $CBC2	55704 $D998	E:(AC)=LB(Anfangs- (YR)=HB adresse)	YR AC	Lädt FAC#2 mit der Konstanten, die ab Anfangsadresse im Speicher steht.
MOVFM	52440 $CED8	55982 $DAAE	E:(AC)=LB(Anfangs- (YR)=HB adresse)	YR AC	Lädt FAC#1 mit der Konstanten, die ab Anfangsadresse im Speicher steht.
VAL1	51435 $C8EB	54927 $D68F	E:($1F)=LB(Anfangs- ($20)=HB(adresse)	AC XR YR	Bringt ASCII-String im Speicher ab der Anfangsadresse nach FAC#1 und wandelt dabei um von Speicher- nach Register-format.
	52493 $CD0D	56035 $DAE3	E:(XR)=LB(Anfangs- (YR)=HB adresse)	AC YR	Ueberträgt FAC#1 in die fünf Speicherstellen ab der Anfangsadresse und wandelt dabei um in Speicher-Format.
MOVFA	52530 $CD32	56072 $DB08	A:($6D)=0(Rundungs- stelle) $66 67 68 69 6A 6B $5E 5F 60 61 62 63	AC XR	Ueberträgt FAC#2 nach FAC#1.
MOVEF	52549 $CD45	56091 $DB1B	A:($6D)=0(Rundungs- stelle) $5E 5F 60 61 62 63 $66 67 68 69 6A 6B	AC XR	Ueberträgt FAC#1 nach FAC#2.
MOVAF	52546 $CD42	56088 $DB18	A:($6D)=0	AC XR	Ueberträgt FAC#1 nach FAC#2 und rundet vorher.

			FAC#1 = ($5E-63) FAC#2 = ($66-6B)		
	52490 $CD2F	56032 $DAE0	E:(XR)=LB(Anfangs- (YR)=HB adresse)	AC YR	Rundet FAC#1, überträgt ihn in die fünf Speicherstellen ab der Anfangsadresse und wandelt dabei um von Register- nach Speicherformat
MOV1F	52480 $CD00	56022 $DAD6	E:(FAC#1) A:($54-58)	AC XR YR	Ueberträgt FAC#1 nach Zwischenspeicher1 und wandelt dabei um von Register- nach Speicherformat.
MOV2F	52477 $CCFD	56019 $DAD3	E:(FAC#1) A:($59-5D)	YR	Ueberträgt FAC#1 nach Zwischenspeicher2 und wandelt dabei um von Register- nach Speicherformat.
MULSHF	51897 $CAB9	55439 $D88F		AC XR YR	Verschiebt Zwischenspeicher1 (=$23-26) um 1 Byte nach rechts.
			$23 24 25 26 65 $23 24 25 26 6D		
	48708 $BE44	52551 $CD47	E:(FAC#1) ($1F)=LB Rück- ($20)=HB sprung- adresse	AC	Schiebt FAC#1 auf den Stapel (Beginn mit $42) und springt dann nach der Adresse, die in ($1F,20) abgelegt ist.
RND	53801 $D229	57215 $DF7F	A:($5E-$63) Zufalls- ($88-8C) zahl	AC XR YR	Erzeugt eine Zufallszahl im FAC#1 und im Zufallszahlenspeicher
	52421 $CC25	55963 $DA9B	E:($23-26) A:($5F-62)	AC	Ueberträgt Register ($23-26) in Mantisse von FAC#1 und macht sie linksbündig.

			FAC#1 = ($5E-63) FAC#2 = ($66-6B)	
		56149 $DB55	E:($5F)=HB Zahl ($60)=LB (XR)=$90 C-Bit = 1 A:($5E-63)=FK-Zahl	AC Wandelt ganze Zahl zwischen XR 0 und 65535 in Fliesskomma- YR zahl um und legt sie in FAC#1 ab.
GIVAYF	50364 $C4BC	53869 $D26D	E:(AC)=HB Zahl (YR)=LB A:($5E-63)=FK-Zahl	AC Wandelt ganze Zahl zwischen XR -32768 und 32767 in Fliess- YR kommazahl um und legt sie in FAC#1 ab.
GETADR	51501 $C92D	54994 $D6D2	E:($5E-63)=FK-Zahl A:($12)=HB Ganze ($11)=LB Zahl	AC Wandelt Fliesskommazahl XR im FAC#1 um in eine ganze YR Zahl zwischen 0 und 65535.
AYINT	59898 $C2EA	53402 $D09A	E:($5E-63)=FK-Zahl A:($61)=HB Ganze ($62)=LB Zahl	AC Wandelt Fliesskommazahl XR im FAC#1 um in eine ganze YR Zahl zwischen -32768 und 32767.
	51479 $C917	54927 $D68F	E:($1F)=LB(Anfangs- ($20)=HB adresse) (AC)=Stringlänge A:($5E-63)=FK-Zahl	AC Wandelt numerischen String XR ab Anfangsadresse um in YR Fliesskommazahl im FAC#1.
FOUT	53139 $CF93	56553 $DCE9	E:($5E-63)=FK-Zahl A:($0100...ff.) V O R S I C H T !	AC Wandelt Fliesskommazahl XR im FAC#1 um in String und YR legt diesen in den Stack- bereich ab. Es muss vorher nachgeprüft werden, ob noch Platz im Stack verfügbar ist!
ASCII	55098 $D73A	59277 $E7D8	E:(AC)= 0...F A:(AC)=ASCII(0...F)	AC Wandelt ein Nibble (=halbes Byte) im AC um in ein ASCII-Zeichen.
HEXIT	55181 $D78D	59360 $E7E0	E:(AC)=ASCII(0...F) A:(AC)= 0...F	AC Wandelt ASCII-Zeichen im AC um in ein Nibble (=halbes Byte)

Arithmetische Routinen Teil A:

FAC#1 = ($5E bis $63)
FAC#2 = ($66 bis $6B)

ZEROFC	51757 $CA2D	55299 $D803		AC XR YR	Löscht FAC#1 durch Nullsetzen des Exponenten und des Vorzeichens.
NEGOP	53579 $D14B	56993 $DEA1		AC	FAC#1 = -FAC#1
FADDH	51583 $C97F	55084 $D72C		AC XR YR	FAC#1 = FAC#1 + 0.5
MUL10	52248 $CC18	55790 $D9EE		AC XR	FAC#1 = FAC#1 * 10
MUL02	52266 $CC2A	55808 $DA00			FAC#1 = FAC#1 * 2
NOTNUL	51621 $C9A5	55163 $D77B		AC XR YR	FAC#1 = FAC#2 + FAC#1
FADD	51613 $C99D	55155 $D773	E:(AC)=LB(Anfangs- (YR)=HB adresse der Konstanten)	AC XR YR	FAC#1 = Konst. + FAC#1
FSUBT	51593 $C989	55094 $D736		AC XR YR	FAC#1 = FAC#2 - FAC#1
FSUB	51590 $C986	55091 $D733	E:(AC)=LB(Anfangs- (YR)=HB adresse der Konstanten)	AC XR YR	FAC#1 = Konst. - FAC#1

		FAC#1 = ($5E bis $63) FAC#2 = ($66 bis $6B)		
NOTZER	52070 55612 $CB66 $D93C		AC XR YR	FAC#1 = FAC#2 * FAC#1
FMULT	52062 55604 $CB5E $D934	E:(AC)=LB(Anfangs- (YR)=HB adresse der Konstanten)	AC XR YR	FAC#1 = Konst. * FAC#1
FDIVT	52296 55840 $CC48 $DA20		AC	FAC#1 = FAC#2 / FAC#1
FDIV	52293 55835 $CC45 $DA1B	E:(AC)=LB(Anfangs- (YR)=HB adresse der Konstanten)	AC XR YR	FAC#1 = Konst. / FAC#1
	53524 56938 $D114 $DE6A		AC XR YR	FAC#1 = FAC#2 ü (FAC#1)
FEXP	53519 56933 $D10F $DE65	E:(AC)=LB(Anfang- (YR)=HB adresse der Konstanten)	AC XR YR	FAC#1 = FAC#2 ü (Konst.)
NORMAL	51725 55267 $CA0D $D7E3	E:(FAC#1) A:(FAC#1)	AC XR	Normalisiert FAC#1 durch Linksschieben der Mantisse bis höchstes Bit=1 und Erniedrigen des Exponenten. 'Overflow Error' bei Bereichsüberschreitung.

Monadische Operationen:

FAC#1 = (\$5E-63)

SGN	52591 56133 \$DB45		AC XR YR	FAC#1 = SGN(FAC#1)
ABS	52622 56164 \$CD8E \$DB64			FAC#1 = ABS(FAC#1)
INT	52738 56280 \$CE02 \$DBD8		AC XR YR	FAC#1 = INT(FAC#1)
SQR	53512 56926 \$D108 \$DE5E		AC XR YR	FAC#1 = SQR(FAC#1)
LOG	52000 55542 \$CB20 \$D8F6		AC XR YR	FAC#1 = LOG(FAC#1)
EXP	53636 57050 \$D184 \$DEDA		AC XR YR	FAC#1 = EXP(FAC#1)
COS	53890 57304 \$D282 \$DFD8		AC XR YR	FAC#1 = COS(FAC#1)
SIN	53897 57311 \$D289 \$DFDF		AC XR YR	FAC#1 = SIN(FAC#1)
TAN	53970 57384 \$D2D2 \$E028		AC XR YR	FAC#1 = TAN(FAC#1)
ATN	54060 57484 \$D32C \$E08C		AC XR YR	FAC#1 = ARCTAN(FAC#1)

FAC#1 = ($5E-63)

SIGN — 52577 56119 / $CD61 $DB37 — AC

E:FAC#1 | >0 | =0 | >0

A: AC=	1	0	$FF
C-Flag=	0	*	1
Z-Flag=	0	1	0
N-Flag=	0	0	1

Ueberprüft FAC#1 auf Vorzeichen und ob =0.

(*:wird nicht verändert!)

FCOMP — 52625 56167 / $CD91 $DB67

E:(AC)=LB(Anfangs-
(YR)=HB adresse)

FAC#1 | >K | =K | <K

A: AC=	1	0	$FF
C-Flag=	0	*	1
Z-Flag=	0	1	0
N-Flag=	0	0	1

($33)=AC, ($34)=YR

Vergleicht FAC#1 mit einer Konstanten (K), die im Speicher ab der Anfangs-adresse steht.

(*:wird nicht verändert!)

52629 56171 / $CD95 $DB6B — E:($33)=LB(Anfangs-($34)=HB adresse) — AC XR YR

Macht dasselbe mit der Konstanten, die im Speicher ab der Anfangsadresse steht.

ISLETC — 49590 53239 / $C1B6 $CFF7 — E:(AC)=Zeichen
A: C-Flag:Buchst.=1
ansonsten=0 — AC

Prüft, ob das Zeichen im AC ein Buchstabe ist.

K o n s t a n t e n i m R O M :

Die hier aufgeführten Konstanten stehen ab der angegebenen Adresse als Gleitkommazahlen im ROM und können für die beschriebenen Mathematik-Routinen verwendet werden (durch Laden von AC und YR mit der Anfangsadresse).

Anfangsadresse		Wert der Konstanten
BASIC 3.0	BASIC 4.0	
57438=$E05E	54024=$D308	+ 0.25
56861=$DE1D	53447=$D0C7	+ 0.5
55532=$D8EC	51990=$CB16	- 0.5
55496=$D8C8	53631=$D17F	+ 1.0
55813=$DA05	52271=$CC2F	+ 10
56511=$DCBF	52969=$CEE9	+ 99999999.9
56516=$DCC4	52974=$CEEE	+ 999999999
56521=$DCC9	52979=$CEF3	+ 1E+09

Weitere wichtige Konstanten:

57428=$E054	54014=$D2FE	+ PI/2
57433=$E059	54055=$D327	+ 2*PI
52643=$CDA3		+ PI
55502=$D8CE	51960=$CAF8	LOG(e)= .4342 (zur Basis 10)
55537=$D8F1	51995=$CB1B	LN(2) = .6931
55527=$D8E7	51985=$CB12	SQR(2)= 1.414
55522=$D8E2	51980=$CB0C	1/SQR(2)= .0707

Zahlendarstellung A:

Die folgende Tabelle enthält eine Auswahl von Zahlen in Speicherformatdarstellung.

.1	7D 4C CC CC CD	01111101 01001100 11001100 11001100 11001101
.2	7E 4C CC CC CD	01111110 01001100 11001100 11001100 11001101
.3	7F 19 99 99 9A	01111111 00011001 10011001 10011001 10011010
.4	7F 4C CC CC CD	01111111 01001100 11001100 11001100 11001101
.5	80 00 00 00 00	10000000 00000000 00000000 00000000 00000000
.6	80 19 99 99 9A	10000000 00011001 10011001 10011001 10011010
.7	80 33 33 33 33	10000000 00110011 00110011 00110011 00110011
.8	80 4C CC CC CD	10000000 01001100 11001100 11001100 11001101
.9	80 66 66 66 66	10000000 01100110 01100110 01100110 01100110
0	00 00 00 00 00	00000000 00000000 00000000 00000000 00000000
1	81 00 00 00 00	10000001 00000000 00000000 00000000 00000000
2	82 00 00 00 00	10000010 00000000 00000000 00000000 00000000
3	82 40 00 00 00	10000010 01000000 00000000 00000000 00000000
4	83 00 00 00 00	10000011 00000000 00000000 00000000 00000000
5	83 20 00 00 00	10000011 00100000 00000000 00000000 00000000
6	83 40 00 00 00	10000011 01000000 00000000 00000000 00000000
7	83 60 00 00 00	10000011 01100000 00000000 00000000 00000000
8	84 00 00 00 00	10000100 00000000 00000000 00000000 00000000
9	84 10 00 00 00	10000100 00010000 00000000 00000000 00000000
10	84 20 00 00 00	10000100 00100000 00000000 00000000 00000000
15	84 70 00 00 00	10000100 01110000 00000000 00000000 00000000
20	85 20 00 00 00	10000101 00100000 00000000 00000000 00000000
25	85 48 00 00 00	10000101 01001000 00000000 00000000 00000000
30	85 70 00 00 00	10000101 01110000 00000000 00000000 00000000
35	86 0C 00 00 00	10000110 00001100 00000000 00000000 00000000
40	86 20 00 00 00	10000110 00100000 00000000 00000000 00000000
45	86 34 00 00 00	10000110 00110100 00000000 00000000 00000000
50	86 48 00 00 00	10000110 01001000 00000000 00000000 00000000

Zahlendarstellung B:

60 / 86 70 00 00 00 / 10000110 01110000 00000000 00000000 00000000

70 / 87 0C 00 00 00 / 10000111 00001100 00000000 00000000 00000000

80 / 87 20 00 00 00 / 10000111 00100000 00000000 00000000 00000000

90 / 87 34 00 00 00 / 10000111 00110100 00000000 00000000 00000000

99 / 87 46 00 00 00 / 10000111 01000110 00000000 00000000 00000000

100 / 87 48 00 00 00 / 10000111 01001000 00000000 00000000 00000000

150 / 88 16 00 00 00 / 10001000 00010110 00000000 00000000 00000000

200 / 88 48 00 00 00 / 10001000 01001000 00000000 00000000 00000000

255 / 88 7F 00 00 00 / 10001000 01111111 00000000 00000000 00000000

300 / 89 16 00 00 00 / 10001001 00010110 00000000 00000000 00000000

400 / 89 48 00 00 00 / 10001001 01001000 00000000 00000000 00000000

500 / 89 7A 00 00 00 / 10001001 01111010 00000000 00000000 00000000

600 / 8A 16 00 00 00 / 10001010 00010110 00000000 00000000 00000000

700 / 8A 2F 00 00 00 / 10001010 00101111 00000000 00000000 00000000

800 / 8A 48 00 00 00 / 10001010 01001000 00000000 00000000 00000000

826 / 8A 4E 80 00 00 / 10001010 01001110 10000000 00000000 00000000

900 / 8A 61 00 00 00 / 10001010 01100001 00000000 00000000 00000000

1000 / 8A 7A 00 00 00 / 10001010 01111010 00000000 00000000 00000000

32768 / 90 00 00 00 00 / 10010000 00000000 00000000 00000000 00000000

65535 / 90 7F FF 00 00 / 10010000 01111111 11111111 00000000 00000000

9999999 / 98 18 96 7F 00 / 10011000 00011000 10010110 01111111 00000000

123456789 / 9B 6B 79 A2 A0 / 10011011 01101011 01111001 10100010 10100000

1.000001 / 81 00 00 08 64 / 10000001 00000000 00000000 00001000 01100100

1.5E-09 / 63 4E 28 8E E3 / 01100011 01001110 00101000 10001110 11100011

1.5E+09 / 9F 32 D0 5E 00 / 10011111 00110010 11010000 01011110 00000000

1E+33 / EE 45 37 19 15 / 11101110 01000101 00110111 00011001 00010101

5E-35 / 0F 04 EC 3C 98 / 00001111 00000100 11101100 00111100 10011000

.25 / 7F 00 00 00 00 / 01111111 00000000 00000000 00000000 00000000

.111111111 / 7D 63 8E 38 E3 / 01111101 01100011 10001110 00111000 11100011

FNDLIN	46499 $B5A3	50476 $C52C	E: ($11)=LB (Zeilen- ($12)=HB nummer) A: ($5C)=LB (Start- ($5D)=HB adresse)	AC XR YR	Sucht eine BASIC-Zeile mit der Zeilennummer und übergibt ihre Startadresse.
	49543 $C187	53193 $CFC9	E: ($42)=1. Zeichen ($43)=2. des Variablen- namens A: ($44)=LB (Adresse ($45)=HB des Variablen- inhaltes)	AC XR YR	Sucht eine einfache Variable und übergibt die Anfangsadresse des Variableninhaltes.
	49987 $C343	53491 $D0F3	E: ($05)=1 ($06)=0 ($42)=1. Zeichen ($43)=2. des Variablen- namens (Stack)=(Index des Feldelemtes HB,LB) A: ($44)=LB (Adresse ($45)=HB des Variablen- inhaltes)	AC XR YR	Sucht in einem eindimensionalen Feld nach dem Feldelement und übergibt dessen Anfangsadresse.

Das COMMODORE-Betriebssystem legt die Basic-Befehle nicht, wie man vielleicht annehmen sollte, als ASCII-Zeichenfolge ab, sondern als sogenannte T O K E N S .
Diese TOKENS sind nichts anderes als 1-Byte-Kodes für die einzelnen Basic-Befehle.
Durch diese Art der Speicherung spart der Rechner natürlich enorm viel Speicherplatz.

128/$80	END	153/$99	PRINT	178/$B2	=
129/$81	FOR	154/$9A	CONT	179/$B3	<
130/$82	NEXT	155/$9B	LIST	180/$B4	SGN
131/$83	DATA	156/$9C	CLR	181/$B5	INT
132/$84	INPUT#	157/$9D	CMD	182/$B6	ABS
133/$85	INPUT	158/$9E	SYS	183/$B7	USR
134/$86	DIM	159/$9F	OPEN	184/$B8	FRE
135/$87	READ	160/$AO	CLOSE	185/$B9	POS
136/$88	LET	161/$A1	GET	186/$BA	SQR
137/$89	GOTO	162/$A2	NEW	187/$BB	RND
138/$8A	RUN	163/$A3	TAB(	188/$BC	LOG
139/$8B	IF	164/$A4	TO	189/$BD	EXP
140/$8C	RESTORE	165/$A5	FN	190/$BE	COS
141/$8D	GOSUB	166/$A6	SPC(	191/$BF	SIN
142/$8E	RETURN	167/$A7	THEN	192/$CO	TAN
143/$8F	REM	168/$A8	NOT	193/$C1	ATN
144/$90	STOP	169/$A9	STEP	194/$C2	PEEK
145/$91	ON	170/$AA	+	195/$C3	LEN
146/$92	WAIT	171/$AB	-	196/$C4	STR$
147/$93	LOAD	172/$AC	*	197/$C5	VAL
148/$94	SAVE	173/$AD	/	198/$C6	ASC
149/$95	VERIFY	174/$AE	ü	199/$C7	CHR$
150/$96	DEF	175/$AF	AND	200/$C8	LEFT$
151/$97	POKE	176/$BO	OR	201/$C9	RIGHT$
152/$98	PRINT#	177/$B1	>	202/$CA	MID$
203/$CB	GO				

Für die Disk-BASIC-Befehle im BASIC 4.0 wurden einige zusätzliche TOKENS eingeführt:

204/$CC	CONCAT
205/$CD	DOPEN
206/$CE	DCLOSE
207/$CF	RECORD
208/$D0	HEADER
209/$D1	COLLECT
210/$D2	BACKUP
211/$D3	COPY
212/$D4	APPEND
213/$D5	DSAVE
214/$D6	DLOAD
215/$D7	CATALOG
216/$D8	RENAME
217/$D9	SCRATCH
218/$DA	DIRECTORY

Anschlussbelegung der Ports:

CBM-Kontakt-Nr.	IEC-Busbelegung	USER-PORT
1	DIO1	LOGIC GND
2	DIO2	TV-VIDEO-DRIVE
3	DIO3	IEEE-SRQ
4	DIO4	IEEE-EOI
5	EOI	DIAGNOSTIC
6	DAV	#1 READ
7	NRFD	#2 READ
8	NDAC	#1,#2 WRITE
9	IFC	TV-VERT.
10	SRQ	TV-HORIZ.
11	ATN	LOGIC GND
12	SHIELD (Abschirmung)	LOGIC GND
A	DIO5	LOGIC GND
B	DIO6	CA1
C	DIO7	DIO1
D	DIO8	DIO2
E	REN	DIO3
F	GND	DIO4
H	GND	DIO5
J	GND	DIO6
K	GND	DIO7
L	GND	DIO8
M	GND	CB2
N	LOGIC GND	LOGIC GND

Die fehlenden Bedeutungen der in dieser Abhandlung nicht aufgeführten Anschlüsse können im COMMODORE-Handbuch nachgelesen werden.

Systemadressen

Label	HEX.	DEZ.	Routine bzw. Speicherstelle
USRPOK	0000	00000	Konstante $4C (JMP-Befehl für USR)
ADDPRC	0001	00001	Adresse für USR: L, H; zeigt standardmässig auf $D123
STRSIZ	0003	00003	Hilfszähler (Startbegrenzer)
ENDCHR	0004	00004	Konstante $00 als Trennfeld (Endbegrenzer)
COUNT	0005	00005	Zeiger in Input-Puffer (Hilfszähler) Anzahl der Dimensionen eines Feldes
DIMFLG	0006	00006	Speicher für 1. Zeichen des Variablennamen bei Dimensionierung
VALTYP	0007	00007	Flag für Variablentyp ($00=numerisch, $FF=String)
INTFLG	0008	00008	Flag für Integer (Integer=$80,sonst =$00)
DORES	0009	00009	Flag für DATA,, FLag für Strings und Remarks, Hochkomma-Flag für LIST
CLMWID	000A	00010	Flag für Dimensionierung bzw. Funktionen
INPFLG	000B	00011	Flag für INPUT ($00), GET/GET# ($40) oder READ ($98)
TANSGN	000C	00012	Flag für das Vorzeichen von TAN und für Vergleichsoperationen
DSDESC	000D	00013	Flag für OUTPUT (negativ, wenn unterdrückt) Länge und Pointer auf ds$
CHANNL	000E	00014	Nummer des aktiven I/O-Kanals
LINNUM	0011	00017	Hilfs-Adresse (für SYS, GOTO, Zeilennummer, ...)
FORSIZ	0012	00018	Anzahl der Bytes, die für FOR-NEXT verwendet werden
TEMPPT	0013	00019	Zeiger in Tabelle der Variablenadressen
LASTPT	0014	00020	Zeiger auf den zuletzt verwendeten String; L, H
TEMPST	0016	00022	Tabelle mit Variablenadressen (Stringvariablen)
NUMLEV	0017	00023	Anzahl erlaubter GOSUBS
INDEX	001F	00031	Transfer-Pointer
INDEX2	0021	00033	Pointer für diverse Zwecke
RESHO	0023	00035	Register für Zwischenspeicherung von Funktionsargumenten
TXTTAB	0028	00040	Zeiger auf den Beginn der BASIC-Befehle, L, H
VARTAB	002A	00042	Zeiger auf den Beginn der Variablentabelle, L, H
ARYTAB	002C	00044	Zeiger auf den Beginn der Array-Tabelle, L, H
STREND	002E	00046	Zeiger auf das Ende der Variablentabelle, L, H
FRETOP	0030	00048	Zeiger auf den Beginn des letzten Strings, L, H
FRESPC	0032	00050	Zeiger auf das Ende des letzten Strings, L, H
MEMSIZ	0034	00052	Zeiger auf die höchste RAM-Adresse+1, L, H
CURLIN	0036	00054	Nummer der gerade ausgeführten Zeile, L, H
DIRBEF	0037	00055	ist $FF, wenn direkter Befehl ausgeführt wird
OLDLIN	0038	00056	lfd. Zeilennummer für CONT, L, H
OLDTXT	003A	00058	Zeiger auf den nächsten Basic Befehl, L,H
DATLIN	003C	00060	Nummer der laufenden DATA-Zeile, L, H
DATPTR	003E	00062	Zeiger auf nächstes DATA, L, H
INPPTR	0040	00064	Zeiger auf die Herkunft des Input, L, H
VARNAM	0042	00066	Name der laufenden Variablen (1.+2. Zeichen)
VARPNT	0044	00068	Adresse der laufenden Variablen, L, H

Label	HEX.	DEZ.	Routine bzw. Speicherstelle
FORPNT	0046	00070	Zeiger auf die lfd. FOR/NEXT-Variable, L, H
EORMSK	0047	00071	Maske für EOR-Verknüpfung bei WAIT
OPPTR	0048	00072	Zeiger auf den lfd. Operator
OPMASK	004A	00074	Maske für den lfd. Operator (< = >)
DEFPNT	004B	00075	Speicherbereich für diverse Zwecke, Zeiger für DEFFN, L, H
DSCPNT	004D	00077	Speicherbereich für garbage collection Zeiger für Stringverarbeitung, L, H
STRLEN	004F	00079	Länge des obigen Strings
BUFLEN	0051	00081	Konstante $4C (JMP für Funktionen)
SIZE	0052	00082	Vektor für Funktionsauswertung Max. Länge+1 des Input-Puffers
TEMPF1	0054	00084	Zwischenspeicher 1 für Funktionsargumente
ARYPNT	0055	00085	Zeiger #1 für Blocktransfer
HIGHTR	0057	00087	Zeiger #2 für Blocktransfer
TEMPF2	0059	00089	Zwischenspeicher 2 für Funktionsargumente
DPTFLG	005C	00092	Zeiger #3 für Blocktransfer
EXPSGN	005D	00093	Zeichen für Exponent zur Basis 10
FACEXP	005E	00094	Fliesskommaakkumulator #1 (FAC#1) Exponent
FACHU	005F	00095	Höchstwertiges Byte der Mantisse
SGNFLG	0064	00100	Zähler für Funktionsauswertung, Vorzeichen von FAC#1
BITS	0065	00101	Zähler für Normalisierung von Akku #1
ARGEXP	0066	00102	Fliesskommaakkumulator #2 (FAC#2) Exponent
ARGSGN	006B	00107	Vorzeichen von ARG (wie beim FAC)
ARISGN	006C	00108	Vorzeichenvergleichsbyte der beiden Akku's
FACOV	006D	00109	Byte für die Rundung von FAC#1 (FAC#2)
POLYPT	006E	00110	Zeiger für Funktionsauswertung und Zeichenkettenverarbeitung, L, H Zeiger in Input-Puffer
CHRGET	0070	00112	CHRGET-Routine, holt nächstes BASIC-Zeichen
CHRGOT	0076	00118	CHRGOT, holt letztes BASIC-Zeichen
TXTPTR	0077	00119	CHRGET-Pointer, L, H
RNDX	0088	00136	enthält nächste Zufallszahl für RND
CTIMR	008D	00141	24-Stunden Uhr in 1/60 Sekunden (Jiffies)
IRQPTR	0090	00144	IRQ-Vektor, L, H, normal $E62E<E455
BRKPTR	0092	00146	Break-Vektor, L, H, normal $FD17<D478
NMIPTR	0094	00148	NMI-Vektor, L, H, normal $C389<B3FF
CSTAT	0096	00150	Status Wort (ST)
TASMAT	0097	00151	Matrixnummer der gedrückten Taste, $FF = keine Taste gedrückt
SHFLG	0098	00152	Flag für Shift, $01 = gedrückt, (sonst $00)
CORFAC	0099	00153	Korrektur-Faktor für Uhr
RVSFLG	009B	00155	Flag für RVS, SPACE, STOP, ., ä, = (PIA), wenn Inhalt = $EF, dann Break
TIMPUF	009C	00156	Puffer für Timing beim Lesen von Kassette
LDFLG	009D	00157	Flag für LOAD (=$00) und Verify (=$01)
TYPNUM	009E	00158	Anzahl der Zeichen im Tastaturpuffer (max. 9), warten auf beliebige Taste: WAIT 158,1
RVSFL2	009F	00159	Flag für RVS-Darstellung auf Bildschirm
IECFLG	00A0	00160	Flag für IEEE-Ausgabe, =$FF wenn noch ein Zeichen auszugeben ist
LINLEN	00A1	00161	Zwischenspeicher für Zeilenlänge (40 oder 80 Zeichen) des Bildschirms
CURSZ	00A3	00163	Zwischenspeicher für Cursor-Zeile
CURSS	00A4	00164	Zwischenspeicher für Cursor-Spalte
IECPUF	00A5	00165	Puffer für Ausgabe auf IEEE-Bus
TASTIN	00A6	00166	Tastatur-Input-Code, solange Taste gedrückt
CURSFL	00A7	00167	Flag für Cursor ein/aus während des Programmablaufes; $00 = Cursor blinkt

Label	HEX.	DEZ.	Routine bzw. Speicherstelle
	00A8	00168	Zähler für Blinkdauer des Cursors
	00A9	00169	Zwischenspeicher für das aktuelle Zeichen unter dem Cursor (Bildschirmcode, kein ASCII)
	00AA	00170	Flag für Cursor ein/aus
	00AB	00171	Flag für EOT-Bit
	00AC	00172	Flag für Input (Bildschirm oder Tastatur)
	00AD	00173	Zwischenspeicher für X-Register
	00AE	00174	Anzahl der offenen Files
	00AF	00175	Nummer des Input-Gerätes ($00 = Tastatur)
	00B0	00176	Nummer des Output-Gerätes ($03 = Bildschirm)
	00B1	00177	Paritätsbyte für Kassettenausgabe
	00B2	00178	Flag: Byte eingelesen
	00B4	00180	Tape-Puffer-Character
	00B5	00181	Zähler für Recorder-Puffer, Pointer bei Filenamen-Transfer
	00B7	00183	serieller Bit-Zähler
	00B9	00185	Zähler für Blocks auf Kassette
	00BA	00186	Count-down für Synchronisationssignal bei Kassettenausgabe
	00BB	00187	Zeiger auf nächstes Zeichen im Tape-Puffer #1
	00BC	00188	Zeiger auf nächstes Zeichen im Tape-Puffer #2
	00BD	00189	Cownt-down für Synchronisationssignal bei Kassetteneingabe Zähler für Schreiben des Vorspanns
	00BE	00190	Flag für Lesefehler von Kassette bzw. Schreiben nächstes Byte
	00BF	00191	Flag für Bit-Lesefehler bzw. Schreiben 1. Bit
	00C0	00192	Zeiger für Korrektur beim Lesen (Pass 1)
	00C1	00193	Zeiger für Korrektur beim Lesen (Pass 2)
	00C2	00194	Flag für Lesen von Kassette, $00=Abtastung, $01-$0F=Cownt-down, $40=Load, $80=Ende
	00C3	00195	Dauer der 'shorts' vor dem Schreiben von Daten, Prüfsummen-Wort
	00C4	00196	Zeiger auf Cursorstellung im Bildschirm-RAM, L, H
	00C6	00198	Cursorposition:Spalte
	00C7	00199	Zeiger für diverse Zwecke, L, H; Start-Adresse für SAVE und LOAD mit IEEE-Bus, (Pointer für Tape-Puffer, Scrolling)
	00C9	00201	End-Adresse für SAVE und LOAD mit IEEE-Bus, Tape-Endadresse
	00CB	00203	Konstanten für Kassetten-Timing
	00CD	00205	Flag für Cursor: direkt=$00 / programmiert<>$00
	00CE	00206	Enable-Flag für Timer 1 (Lesen von Kassette), wenn =$00,dann Timer nicht in Funktion
	00CF	00207	Flag für den Empfang von EOT von Kassette
	00D0	00208	Character Lesefehler
	00D1	00209	Länge des lfd. Filenamen
	00D2	00210	Logische Nummer des laufenden Files
	00D3	00211	Sekundäradresse des lfd. Files
	00D4	00212	Primäradresse des lfd. Files
	00D5	00213	Flag für einfache ($27) oder doppelte ($4F) Zeile BASIC 4.0: Letzte Spalte des Bildschirm-Fensters
	00D6	00214	Zeiger auf den laufenden Tape-Puffer
	00D8	00216	Cursorposition Zeile (0-24).

Label	HEX.	DEZ.	Routine bzw. Speicherstelle
	00D9	00217	Zwischenspeicher für Input/Output (letzte Taste (ASCII), Puffer-Prüfsumme)
	00DA	00218	Zeiger auf den laufenden Filenamen
	00DC	00220	Anzahl der Inserts (Taste)
	00DD	00221	Serielles Bit-Shift-Wort (Schreiben/Lesen)
	00DE	00222	Anzahl der noch zu lesenden/schreibenden Blocks
	00DF	00223	Serieller Wort-Puffer
	00E0	00224	BASIC 3.0:Tabelle der MSB's der Zeilenanfänge
			---------- (Bildschirm) bis 248
	00E0	00224	BASIC 4.0: Anfangszeile d. Bildschirmfensters
	00E1	00225	---------- Endzeile d. "
	00E2	00226	Anfangs-Spalte d. "
	00E3	00227	Länge des Tastaturpuffers
	00E4	00228	REP-Flag (Bit 7 relevant)
	00E5	00229	REP-Zähler 2
	00E6	00230	REP-Zähler 1
	00E7	00231	Zeitkontante für die Glocke
	00E8	00232	HOME-Flag (wird nach 1.HOME gesetzt, nach 2.HOME oder anderem Zeichen gelöscht)
	00E9	00233	Vektor für indir. Sprung von 57626 nach 57629
	00EB	00235	Vektor für indir. Sprung von 57865 nach 57868
	00F9	00249	Interrupt-Flag für Recorder-Schalter #1, $00 bedeutet, dass keine Taste gedrückt ist
	00FA	00250	Interrupt-Flag für Recorder-Schalter #2 $00 bedeutet, dass keine Taste gedrückt ist
	00FB	00251	Zeiger für Betriebssystem (LOAD, VERIFY) (Startadresse für LOAD)
	0100	00256	Hardware-Stack
FBUFFR	0100	00256	Puffer für Binär- nach ASCII-Konvertierung
	0100	00256	Hilfsspeicher für Fehlerkorrektur bei Bandeingabe
	013E	00318	Speicher im Zusammenhang mit GOSUB und FOR-NEXT
STKEND	01FB	00507	Ende des Stacks für BASIC
BUF	0200	00512	BASIC-Input-Puffer
			Programmzähler PC: H, L
	0202	00514	Prozessor Status SR
	0203	00515	Akkumulator AC
	0204	00516	X-Register XR
	0205	00517	Y-Register YR
	0206	00518	Stack-Pointer SP
	0207	00519	IRQ-Vektor für externen Interrupt: H, L
	0251	00593	Tabelle der logischen File-Nummern
	025B	00603	Tabelle der Primäradressen der Files (Geräte-Adressen)
	0265	00613	Tabelle der Sekundäradressen der Files

Label	HEX.	DEZ.	Routine bzw. Speicherstelle
	026F	00623	Tastatur-Input-Puffer
	0279	00633	Prüfsumme für Tape Lesen
	027A	00634	Puffer für Recorder #1
	027A	00634	Typ des Datenblocks: 1: Programmfile-Header (SAVE, LOAD) 2: Datenblock (PRINT#, INPUT#, GET#) 4: Datenfile-Header (OPEN)
	027B	00635	Startadresse LB, HB des Programmes (SAVE, LOAD) des Kassettenpuffers (OPEN)
	027D	00637	Endadresse LB, HB des Programmes (SAVE, LOAD) des Kassettenpuffers (OPEN)
	033A	00826	Puffer für Recorder #2
	033A	00826	Typ des Datenblocks: 1: Programmfile-Header (SAVE, LOAD) 2: Datenblock (PRINT#, INPUT#, GET#) 4: Datenfile-Header (OPEN)
	033B	00827	Startadresse LB, HB des Programmes (SAVE, LOAD) des Kassettenpuffers (OPEN)
	033D	00829	Endadresse LB, HB des Programmes (SAVE, LOAD) des Kassettenpuffers (OPEN)
	03FA	01018	Vektor für Monitor, zeigt normalerweise auf $E7F7 < D7A4, kann vom Benutzer verbogen werden für eigene Monitor-Befehle
RAMLOC	0400	01024	Beginn des freien RAM-Bereiches
	8000	32768	Video-RAM
	9000	36864	Raum für ROM/RAM-Erweiterungen RAM-Bereich nicht für BASIC-Programme nutzbar, wohl aber nach Pointer-Veränderung für Variablen des BASIC-Programmes

Label	3.0	4.0	Routine bzw. Speicherstelle
STMDSP	C000	B000	Adressentabelle der BASIC-Routinen
RESLST	C092	B0B2	BASIC-Befehlstabelle
ERRTAB	C192	B20D	Tabelle der Fehlermeldungen (Text siehe unten)
	C192	B20D	'NEXT WITHOUT FOR', XR=$00
	C1A2		'SYNTAX', XR=$10
	C1A8		'RETURN WITHOUT GOSUB', XR=$16
	C1BC		'OUT OF DATA', XR=$2A
	C1C7		'ILLEGAL QUANTITY', XR=$35
	C1D7		'OVERFLOW', XR=$45
	C1DF		'OUT OF MEMORY', XR=$4D
	C1CE		'UNDEF'D STATEMENT', XR=$5A
	C1FD		'BAD SUSCRIPT', XR=$6B
	C20A		'REDIM'D ARRAY', XR=$78
	C217		'DIVISION BY ZERO', XR=$85
	C227		'ILLEGAL DIREKT', XR=$95
	C235		'TYPE MISMATCH', XR=$A3
	C242		'STRING TOO LONG', XR=$B0
	C251		'FILE DATA', XR=$BF
	C25A		'FORMULA TOO COMPLEX', XR=$C8
	C26D		'CAN'T CONTINUE', XR=$DB
	C27B		'UNDEF'D FUNCTION', XR=$E9
ERR	C28B	B306	'ERROR' bzw. 'ERROR IN'
INTXT	C292	B30D	'IN'
READY	C297	B312	'READY.'
BRKTXT	C2A2	B31B	'BREAK'
FNDFOR	C2AA	B322	Sucht im Stack nach der nächsten bzw. gewünschten FOR-Loop
BLTU	C2D8	B350	Raum für Zeileneinschiebung im BASIC-Programm schaffen
BLTUC	C2DF	B357	Speicherblock-Transfer
GETSTK	C31B	B393	überprüfung, ob Platz im Stapel wenn nicht, Fehlermeldung: OUT OF MEMORY ERROR
REASON	C328	B3A0	Prüfung, ob benöt. Platz im Speicher vorhanden: bei Kollision BASIC und Strings Entfernung nicht benötigter Strings veranlassen, gegebenenfalls 'OUT OF MEMORY ERROR' ausgeben
OMERR	C355	B3CD	Behandelt alle Fehlerausdrucke X-indiziert (s.o.), Einsprung ERROR, sonst '?OUT OF MEMORY ERROR', springt dann nach READY.
ERROR	C357	B3CF	Error-Behandlung (Fehler in XR)
	C368		'OUT OF MEMORY ERROR' ausgeben
READY	C389	B3FF	BASIC-Warm-Start, gibt READY aus
MAIN	C392	B406	Holen, Analysieren und Ausführen einer Zeile

Label	3.0	4.0	Routine bzw. Speicherstelle
MAIN1	C3AB	B41F	Verarbeitet neue BASIC-Zeilen, zerstört alte BASIC-Zeilen
	C3B8		Zeile löschen
	C407		Zeile einfügen
FINI	C439	B4AD	BASIC-System ordnen
LNKPRG	C442	B4B6	Bei Insert/Delete Programmzeilen neu binden
INLIN	C46F	B4E2	Zeichen von Tastatur in den Eingabe-Puffer holen
	C481	B4FB	Zeichenübernahme vom aktiven Kanal
CRUNCH	C495	B4FB	Umwandlung des Textes im Eingabepuffer in den BASIC-Kode
FNDLIN	C52C	B5A3	Sucht nach BASIC-Zeilennummer mit gegebener Zeilennummer
SCRATH	C55B	B5D2	NEW-Befehl, alles löschen (Variable, etc.), direkter Einsprung: SCRATH+2
CLEAR	C577	B5EE	CLR-Befehl, direkter Einsprung: CLEAR+2
STXTPT	C5A7	B622	CHRGET-Pointer auf Programmanfang setzen
LIST	C5B5	B630	LIST-Befehl, Anfangsadresse in $5C/5D, letzte zu listende Zeile in $11/12
FOR	C658	B6DE	Führt eine FOR-NEXT-Schleife aus
NEWSTT	C6C4	B74A	Interpreter-Schleife nächsten Befehl holen und ausführen, nächste Zeile suchen
RUNBAS	C6DA	B75F	RUN BASIC-Programm
GONE3	C700	B785	BASIC-Befehl als Subroutine ausführen
SNERR1	C71E	B7A9	SYNTAX-Error-Vektor
RESTOR	C730	B7B7	RESTORE-Befehl
STOP	C73F	B7C6	STOP, END, und BREAK ausführen
CONT	C76B	B7EE	CONT-Befehl, Zeiger auf nächsten Befehl in $3A/3B, Einsprung: CONT+2
RUN	C785	B808	RUN-Befehl, direkter Einsprung: RUN+2
GOSUB	C790	B813	GOSUB-Befehl
GOTO	C7AD	B830	GOTO-Befehl
RETURN	C7DA	B85D	RETURN-Befehl
	C7E8	B865	'RETURN WITHOUT GOSUB ERROR' ausgeben
USERR	C7EB	B86E	'BAD SUBSCRIPT ERROR' -Vektor
SNERR2	C7F0	B873	Syntax-Error-Vektor einstellen
ERTU1	C7F3	B876	DATA-Befehl
DATAN	C80E	B891	Nächsten BASIC-Befehl suchen, Anzahl der überlesenen Bytes im Y-Register
REMN	C811	B494	Sucht den Anfang der nächsten BASIC-Zeile, Anzahl der überlesenen Bytes im Y-Register
IF	C830	B8B3	IF-Befehl, wenn Exponent von FAC#1 ($5E) =$00, so ist die Abfrage nicht erfüllt
REM	C843	B8C6	REM-Befehl
ONGOTO	C853	B8D6	ON-Befehl
SNERR3	C85B	B8DE	Syntax-Error Vektor
LINGET	C873	B8F6	Umwandlung eines Zeilennummer-Strings in den 2-Byte-Kode, in dem die Zeilennummer im Programmspeicher abgelegt ist
LET	C8AD	B930	LET-Befehl
TIMNUM	C928	B9AB	Prüft numerisches Zeichen, erhöht String-Zeiger; ein ASCII-Zeichen einlesen, zu FAC#1 addieren
FCERR2	C92F	B9B2	ILLEGAL QUANTITY ERROR -Vektor
GETSPT	C937	B9BA	Kopiert Strings, wenn benötigt
STRADJ		BA4E	Zeigt auf zu kopierenden String
PRINTN	C98B	BA88	PRINT#-Befehl
CMD	C991	BA8E	CMD-Befehl
PRINT	C9A5	BAA2	PRINT-Befehl
CRDO	C9E2	BADF	Gibt CARRIAGE RETURN aus
TABER	C9FC	BAFD	Ausführung von TAB und SPC
STROUT	CA1C	BB1D	Druckt einen String, dessen Anfangsadresse sich in A/Y (L,H) befindet und der durch $00 beendet wird

Label	3.0	4.0	Routine bzw. Speicherstelle
OUTSPC	CA39	BB3A	Space ausgeben
CRTSKP	CA40	BB41	Cursor rechts ausgeben
OUTQST	CA43	BB44	'?' ausgeben
OUTDO	CA45	BB46	Character auf aktiven Kanal ausgeben
TRMNOK	CA4F	BB4C	Verarbeitet unzulässige Eingabedaten
	CA68		'FILE DATA ERROR' ausgeben
	CA6D	BB6A	'REDO FROM START' ausgeben
GET	CA7D	BB7A	GET-Befehl
INPUTN	CAA7	BBA4	Input#-Befehl
IODONE	CAB7	BBB4	Eingabe auf Tastatur zurücksetzen
INPUT	CAC1	BBBE	Input-Befehl
QINLIN	CAFA	BBF5	Input veranlassen und verarbeiten
READ	CB07	BC02	READ-Befehl
VARYO	CBEA	BCE5	'EXTRA IGNORED' ausgeben
EXIGNT	CBFC	BCF7	Text: 'EXTRA IGNORED' und 'REDO FROM START'
TRYAGN	CC0D	BD07	'? REDO FROM START'
NEXT	CC20	BD19	NEXT-Befehl
	CC32		'NEXT WITHOUT FOR ERROR' ausgeben
LOOPDN	CC79	BD72	BASIC- und DATA-FORMAT überprüfen, TYPE MISMATCH veranlassen
FRMNUM	CC8B	BD84	Auswerten jedes Terms, dessen Anfangsadresse in $77/78 (L,H) steht (z.B. ASCII nach binär)
CHKNUM	CC8E	BD87	Wenn laufende Variable nicht numerisch, 'TYPE MISMATCH ERROR' ausgeben
CHKSTR	CC90	BD89	Wenn laufende Variable keine Stringvariable, 'TYPE MISMATCH ERROR' ausgeben
CHKERR	CC9A	BD93	'TYPE MISMATCH ERROR' ausgeben
FRMEVL	CC9F	BD98	Input und Auswerten jedes Terms (numerisch und String); bei Aufruf mit SYS FRMEVL (String-variable) steht in $61/62 (L,H) eine Adresse, unter der zuerst die Länge und dann die Adresse (L,H) des betreffenden Strings steht. Bei Aufruf mit einer numerischen Variablen steht in Fliesskommaakku 1 die entsprechende Zahl in Gleitkommadarstellung; Zeiger auf den Term: $77/78
DOPRE1	CD21	BE1A	Zwischenspeicherung von Argumenten auf dem Stack, z.B. FOR-NEXT-Parameter
PULSTK	CD67	BE64	ARG vom STACK zurückholen
EVAL	CD84	BE81	Auswerten von Konstanten, Variablen, PI, Identifizieren von Symbolen
PIVAL	CDA3	BEA0	Wert von PI in Gleitkommadarstellung
QDOT	CDA8	BEA5	überprüft Statements auf +, −, ., E und Hochkomma
NOTOP	CDCF	BECC	NOT-Befehl
EVAL4	CDDE	BEDB	überprüfung der verschiedenen Funktionen (s. auch: ISFUN)
PARCHK	CDEC	BEE9	Prüfen und Auswerten eines arithmetischen Terms in Klammern, CHRGET-Zeiger ($77/78) zeigt hierbei auf die linke Klammer, bei Fehler Sprung nach SNERR
CHKCLS	CDF2	BEEF	Prüft laufendes BASIC-Zeichen (Zeiger $77/78) auf rechte Klammer ($29)
CHKOPN	CDF5	BEF2	Prüft laufendes BASIC-Zeichen (Zeiger $77/78) auf linke Klammer ($28)
CHKCOM	CDF8	BEF5	Prüft laufendes BASIC-Zeichen (Zeiger $77/78) auf Komma ($2C)
SYNCHR	CDFA	BEF7	Vergleicht TXTPTR und AC auf Ungleichheit
SNERR	CE03	BF00	'SYNTAX ERROR' ausgeben (Sprung in Fehlerroutine)

LABEL	3.0	4.0	Routine bzw. Speicherstelle
DOMIN	CE08	BF05	Prüft BASIC-Statement vor der Ausführung auf Fehler(z.B. Klammern bei Funktionen)
ISVAR	CE0F	BF8C	Beginnt Suche nach einer Variablen
	CE26		Ueberprüfung der Systemvariablen TI, TI$, ST
ISVDS		BFC1	Variable ds$ testen und behandeln
CHKDS		BFFC	Sucht nach einer ds-Variablen
GETTIM	CE69	C003	Weist die Zeit der Variablen ti zu
ISFUN	CE89	C047	Art der Funktion überprüfen
OROP	CEC8	C086	Ausführung von 'OR'
ANDOP	CECB	C089	Ausführung von 'AND'
DOREL	CEF8	C0B6	Ausführung von Vergleichsoperationen < = >
DIM3	CF60	C11E	DIM-Befehl
DIM	C121	CF63	DIM-Routine
PTRGET	CF6D	C12B	Variablennamen auswerten, zugehörige Adresse finden
	CFC9	C187	Sucht nach einer Variablen
	CFF7	C1B6	Prüfung, ob 1. Variablenzeichen ein Buchstabe: wenn ja, Carry=1, sonst 0
NOTFNS	D001	C1C0	Speicherplatz für neue Variable einrichten, Name befindet sich dann in $42/43, die Adresse in $44/45
QSTAVR	D01F	C1DE	Sucht Variable st
QDSVAR		C1E6	Sucht Variable ds
	D058		Variable (Zeiger $5C/5D) = 0 setzen
	D069		Speichert Zeiger auf die gefundene/neue Variable nach $44/45
FMAPTR	D078	C2C8	Berechnet Adressen von Feldelementen
N32768	D089	C2D9	Fliesskommakonstante -32768
INTIDX	D08D	C2DD	BASIC-Zahl holen (Zeiger $77/78) und mit Vorzeichen nach Integer konvertieren
AYINT	D09A	C2EA	Gleitkommazahl in 16 Bit Integer (in $61/62; HB/LB)
ISARY	D0AC	C2FC	Suchen bzw. Einrichten von Feldern
BSERR	D120	C370	'BAD SUBSCRIPT ERROR' ausgeben
FCERR	D123	C373	'ILLEGAL QUANTITY ERROR' ausgeben
UMMULT	D228	C477	Integer-Arithmetikroutine für mehrdim. Felder
FRE	D259	C4A8	FRE-Befehl
GIVAYF	D26D	C4BC	16 Bit Integer (in AC/YR) (LB,HB) nach Gleitkommazahl konvertieren (mit Vorzeichen)
POS	D27A	C4C9	POS-Befehl
ERRDIR	D280	C4CF	Prüft auf direkte Kommandos, gibt ggfs. 'ILLEGAL DIRECT' aus
ERRGUF	D288	C4D7	'UNDEF'D FUNCTION ERROR' ausgeben
DEF	D28D	C4DC	Verarbeitet DEF-Kommandos und FN-Funktionen
GETFNM	D2BB	C50A	FN-Syntax prüfen
FNDOER	D2CE	C51D	FN auswerten
STRD	D33F	C58E	STR$-Befehl
STRINI	D34F	C59E	Zeiger auf String berechnen
STRLIT	D361	C5B0	Sucht nach Elementen von Zeichenketten
GETSPA	D3CE	C61D	Zeiger auf Zeichenkette berechnen ($30/31, $32/33)
GARBA2	D400	C66A	Entfernt nicht benötigte Strings (GARBAGE COLLECT)
GRBEND		C700	Sprung nach GARBAGE-COLLECT-Ende
	D497		Sucht nach dem am besten zu löschenden String
	D4E0		String löschen
CAT	D517	C74F	Stringverkettung durchführen
	D532		'STRING TOO LONG ERROR' ausgeben
MOVINS	D554	C78C	String in Speicher übernehmen
MOVSTR	D562	C79A	String in Speicher übernehmen, Anfang in $1F/20, Stringlänge in YR, Transferadresse in $32/33

LABEL	3.0	4.0	Routine bzw. Speicherstelle
FRESTR	D57D	C7B5	Stringadresse und Länge nach Argument-auswertung ermitteln
FRETMP	D584	C7BC	Setzt aktuelle Stringpointer frei
FRETMS	D5B5	C811	Descriptorentabelle nach Löschen anpassen
CHRD	D5C6	C822	CHR$-Befehl
LEFTD	D5DA	C836	LEFT$-Befehl
RIGHTD	D606	C862	Right$-Befehl
MIDD	D611	C86D	MID$-Befehl
PREAM	D63B	C897	Parameter für Stringverarbeitungsfunktionen vom Stack holen
LEN	D656	C8B2	LEN-Befehl
ASC	D665	C8C1	ASC-Befehl
GTBYTC	D675	C8D1	Holt ein Byte aus dem BASIC-Text ins X-Register
VAL	D687	C8E3	VAL-Befehl
STRFKZ	D68F	C8EB	Umwandlung String in eine reelle Zahl
GETNUM	D6C6	C921	Liest zwei Zahlen aus BASIC-Text (16 Bit und 8 Bit, z.B. für POKE), Zeiger $77/78
GETADR	D6D2	C92D	Zahl in Fliesskommaakku #1 in 16 Bit Integer konvertieren und nach $11/12 speichern (überprüft, ob Zahl zwischen 0 und 65535)
PEEK	D6E8	C943	PEEK-Befehl
POKE	D707	C95A	POKE-Befehl
FNWAIT	D710	C963	WAIT-Befehl, warten bis (adressiertes Byte EOR 2. Parameter) AND 1. Parameter <>$00
FADDH	D72C	C97F	Addiert 0.5 zu Akku #1
FSUB	D733	C986	Variable (Zeiger AC/YR) minus Akku #1
FSUBT	D736	C989	Akku #2 minus Akku #1
STORD1	D745		'MICROSOFT!' auf Bildschirm bringen
FADD	D773	C99D	Variable (Zeiger AC/YR) plus Akku #1
ADFACS	D77B	C9A5	Akku #2 plus Akku #1
ZEROFC	D803	CA2D	FAC = 0 setzen
ZEROML	D807	CA31	Vorzeichen von FAC positiv setzen
NORM2	D829	CA53	Normalisierung von Akku #1
NEGFAC	D853	CA7D	Komplementiert Akku #1
INCFAC	D87B	CAA5	Inkrementiert FAC
OVERR	D88A	CAB4	'OVERFLOW ERROR' ausgeben
MULSHF	D88F	CAB9	Routine für Byteweise Multiplikation
FONE	D8C8	CAF2	Fliesskommakonstanten für LOG
LOG	D8F6	CB20	LOG-Befehl
FMULT	D934	CB5E	Multipliziert Variable (Zeiger in AC/YR) mit Akku #1
FMULTT	D937	CB61	Multipliziert Akku #2 mit Akku #1
MULFAC	D93C	CB66	
MLTPLY	D965	CB8F	Bitweise Multiplikation
CONPUK	D998	CBC2	FAC#2 mit Konstante laden
MULDIV	D9C3	CBED	Korrektur von Akku #1 und Akku #2
MLDVEX	D9E0	CC0A	Behandelt Underflow und Overflow
MUL10	D9EE	CC18	Multiplikation: FAC#1 * 10
MUL02	DA00	CC2A	Multiplikation: FAC#1 * 2
TENC	DA05	CC2F	Fliesskommakonstante 10
DIV10	DA0A	CC34	Division: FAC#1 / 10
FDIV	DA1B	CC45	Division Variable (Zeiger AC/YR) durch Akku #1
FDIVT	DA1E	CC48	Division Akku #2 durch Akku #1
DV0ERR	DA96	CCC0	'DIVISION BY ZERO ERROR' ausgeben
MOVFM	DAAE	CCD8	FAC#1 mit Konstante laden
MOV2F	DAD3	CCFD	Akku #1 nach $59 ... $5D bringen
MOV1F	DAD6	CD00	Akku #1 nach $54 ... $58 bringen
MOVVF	DADC	CD06	Akku #1 nach laufender FOR-NEXT-Variable bringen
MOVMF	DAE0	CD0A	Akku #1 in Speicher übertragen mit Rundung
	DAE3	CD0D	Akku #1 in Speicher übertragen ohne Rundung

LABEL	3.0	4.0	Routine bzw. Speicherstelle
MOVFA	DB08	CD32	Akku #2 nach Akku #1 kopieren
MOVAF	DB18	CD42	Akku #1 nach Akku #2 kopieren mit Rundung
MOVEF	DB1B	CD45	Akku #1 nach Akku #2 kopieren ohne Rundung
ROUND	DB27	CD51	Akku #1 aufrunden
SIGN	DB37	CD61	Vorzeichen von Akku #1 feststellen
SGN	DB45	CD6F	SGN-Befehl
FLOATC	DB55	CD7F	Umwandlung Adresse in reelle Zahl
ABS	DB64	CD8E	ABS-Befehl
FCOMP	DB67	CD91	BASIC-Variable mit Akku #1 vergleichen, Zeiger $21/22
QUINT	DBA7	CDD1	Konvertierung von Gleitkomma nach Integer (ohne Vorzeichen)
INT	DBD8	CE02	INT-Befehl
CLRFAC	DBF5	CE1F	Setzt den Wert im AC auf alle Stellen des FAC (z.B.: FAC löschen)
FIN	DBFF	CE29	Umwandlung eines numerischen Strings in eine Flieskommazahl
	DC8A		Liest ASCII-Zeichen ('0' ... '9') aus BASIC-Text
NO999	DCBF	CEE9	Konstantentabelle für Konvertierung Fliess-komma nach String
INPRT	DCCE	CF78	Ausgabe einer Zeilennummer (bei Fehlermeldungen)
LINPRT	DCD9	CF83	16 Bit Integer (in XR/AC LB,HB) dezimal auf den Bildschirm ausgeben
	DCE3	CF93	Fliesskommaakku #1 ausgeben
FOUT	DCE9	CF93	Konvertiert Gleitkommazahl ab $5E nach ASCII und legt den entstehenden String ab $0100 ab
	DE1D	D0C7	Umwandlungskonstanten
FHALF	DE1D	D0C7	Fliesskommakonstante 0.5 für SQR
FOUTBL	DE22	D0CC	Tabelle für Exponentberechnung: 10 hoch x
FDCENQ	DE46	D0F0	Tabellenende
SQR	DE5E	D108	SRQ-Befehl
FEXP	DE65	D10F	FAC#1 = FAC#2 hoch Konstante
FPWRT	DE68	D112	Berechnung der Potenzfunktion; Argument in Akku #2, Exponent in Akku #1
NEGOP	DEA1	D14B	Vorzeichenwechsel von FAC#1
LOGEB2	DEAC	D156	Fliesskommakonstanten für EXP
EXP	DEDA	D184	EXP-Befehl
POLYX	DF2D	D1D73	Wertet eine Folge von Funktionen aus
POLY	DF43	D1ED	Polynomauswertung
RMULC	DF77	D221	Konstanten für RND
RND	DF7F	D229	RND-Befehl
COS	DFD8	D282	Ausführung von 'COS'
SIN	DFDF	D289	Ausführung von 'SIN'
TAN	E028	D2D2	Ausführung von 'TAN'
PI2	E054	D2FE	Fliesskommakonstanten für SIN, COS, TAN
ATN	E08C	D32C	Ausführung von 'ATN'
ATNCON	E0BC	D35C	Fliesskommakonstanten für ATN
INITAT	E0F9	D399	Kopie der CHRGET-Routine, wird übertragen nach $0070-$0087
CHDRTS	E110	D3B0	Anfangswert für RND
INIT	E116	D3B6	CHRGET herstellen, erreichbaren RAM-Bereich feststellen, Stackpointer = $FF, JMP FCERR nach $00-$03, Pointer herstellen ($28-$35), NEW, Ausgabe der Kopfzeile und 'BYTES FREE'
WORDS	E1B7	D44B	Text: 'BYTES FREE' und '*** COMMODORE BASIC ***' bzw. 'BASIC 4.0'
FREMES	E1C4	D458	Meldung ab (WORDS) ausgeben
LASTWR	E1DE	D472	Initialisierung (Stack, Register, I/O ...)
	E229		Bildschirm löschen
	E257		Cursor home
	E25D		Cursorposition ($C4/C5) und Spalte ($C6) neu setzen (auf Zeilenanfang).

LABEL	3.0	4.0	Routine bzw. Speicherstelle
	E285	E0A7	Nächstes Zeichen aus Tastatur-Puffer holen, Pufferzeiger $9E um 1 erniedrigen
	E29A	E0BC	Warteschleife für Tastendruck, Bildaufbereitung
	E2F4	E116	Eingabe vom Bildschirm
	E33F	E16A	Umschaltung Cursor direkt/programmiert und umgekehrt
	E34C	E177	Bildschirmverwaltung für Ausgabe
	E38B		Zeile auf 80 Zeichen begrenzen
	E396		40-Zeichen-Zeile auf 80 Zeichen ausdehnen
	E39C		Display nach oben rollen
	E3B4		Rückschritt in vorhergehende Zeile
	E3D8	E202	Video-Display-Routine, einen Character ausgeben
	E498		Insert-Mode
	E519		Eine Leerzeile ausgeben, (RVS, Cursorprogrammierung, Zeilenlänge usw. werden nicht geändert!) wenn nötig, wird der Bildschirminhalt nach oben verschoben
NXTLIN	E52F	E3B6	Leerzeile ausgeben
	E53F		Vollen Bildschirm eine oder zwei Zeilen hoch (je nach Zeilenlänge, Cursor in die nächste Zeile)
	E5BA		Auf Bildschirm neue Zeile durch Oeffnen des Textes einrichten (als Folge von Insert)
	E61B		Interrupt-Routine
INTRPT	E62E	E455	Uhr-Steuerung, Tastatur-Scanning (1/60 s); wird der Interrupt-Vektor auf INTRPT+3 abgeändert, so wird die Stop-Taste nicht mehr abgefragt (Uhr steht dann!), Aenderung auf INTRPT+31 bewirkt, dass zusätzlich der Cursor nicht mehr blinkt
	E6E4		Rückkehr vom Interrupt Register vom Stack holen (AC, XR, YR, Status)
	E6EA		Speichern eines Zeichens im Akku an momentane Kursorposition im Bildschirm-RAM während der Dunkelpause
	E6F8		Tastatur-Dekodiertabelle
	E748		Tabelle der niederwertigen Bytes der Bildschirmzeilenanfänge
	E761		Bytes für 'LOAD RETURN' und 'RUN RETURN', werden bei Shift RUN/STOP in den Tastatur-Puffer geholt
	E76A	D717	Ausgabe von hexadezimalen Adressen (Monitor)
	E775	D722	1 Hex-Byte ausgeben
	E784		ASCII-Zeichen aus XR und Akku ausgeben
	E797	D744	Austausch von TMP0 und TMP2 (Zeiger des Monitors $FB <-> $FD, $FC <-> $FE)
HADRIN	E7A7	D754	Eingabe hexadezimaler Adressen (Monitor), Speicherung nach $FB/FC (LB, HB)
HEXINP	E7B6	D763	Eliminierung von max. 2 Blanks (Monitor), 1 Hex-Byte einlesen
	E7CB	D778	Zusammenfassung von zwei Zeichen zu einem Byte (Monitor)
	E7E0	D78D	Zeichen für die Kombination zu einem Byte vorbereiten (Monitor)
	E7EB	D798	Eingaberoutine des Monitors, bei Return zur Befehlsabfrageroutine des Monitors, sonst zur aufrufenden Routine
	E7F7	D7A4	Ausgabe von '?', Error-Routine des Monitors

I D E N T I S C H für alle COMMODORE- Rechner:

HEX.	DEZ.	Routine bzw. Speicherstelle
E800	59392	I/O Ports und Expansion I/O
E810	59408	6520 PIA, Adressen E810-E813, Position C7
E810	59408	PIAL, I/O Port A und Datenrichtungsregister Bit 0-3 (Output): Tastaturansteuerung Bit 4 (Input): Schalter Rekorder #1 Bit 5 (Input): Schalter Rekorder #2 Bit 6 (Input): EOI Signal vom IEEE-Bus EOI = End or identify Bit 7 (Input): Abfrage auf Testpin (User Port) WAIT 59408,16,16 wartet, bis Tasten an Rekorder #1 gedrückt werden WAIT 59408,16 wartet, bis alle Tasten an Rekorder #1 ausgerastet werden (32 für Rekorder #2)
E811	59409	PIAL, Kontroll-Register A
E812	59410	PIAK, I/O Port B und Datenrichtungsregister Bit 0-7 (Input): Eingabe von Tastaturabfrage enthält normalerweise $FF $FF = keine Taste, $FE = 0./1. Spalte, $FD = 2./3. Spalte, $FB = 4./5. Spalte $F7 = 6./7. Spalte, $EF = 8./9. Spalte $DF = 10. Spalte, $BF = 11./12. Spalte $7F = 13./14. Spalte; werden mehrere Tasten gedrückt, so ergibt sich der abgefragte Wert aus der UND-Verknüpfung der zugehoerigen Werte. Wait 59410,4,4 wartet, bis die Leertaste gedrückt wird.
E813	59411	PIAS, Kontroll-Register B Bit 3 (Output): Motorkontrolle Rekorder #1 Poke 59411,53: Motor ein, 61: aus
E820	59424	6520 PIA, Adressen E820-E823 Position C6
E820	59424	IEEI, I/O Port A und Datenrichtungsregister Bit 0-7 (Input): DI1-DI8 Input vom IEEE-Bus Einlesen vom IEEE-Bus: PEEK(59424)
E821	59425	IEEIS, Kontroll-Register A Bit 3 (Output): NDAC Signal zum IEEE-Bus (CA2) NDAC = Not data accepted NDAC low: POKE 59425,52 NDAC high: POKE 59425,60
E822	59426	IEEO, I/O Port B und Datenrichtungsregister Bit 0-7 (Output): DO1-DO8 Output zum IEEE-Bus Schreiben auf IEEE-Bus: POKE 59426,data
E823	59427	IEEOS, Kontroll-Register B Bit 3 (Output): DAV Signal zum IEEE-Bus (CB2) DAV = Data valid Bit 4 (Input): SRQ Signal vom IEEE-Bus (CB1) SRQ = Service request

HEX.	DEZ.	Routine bzw. Speicherstelle
E840	59456	6522 VIA, Adressen E840-E84F Position C5
E840	59456	PIA, I/O Port B Bit 0 (Input): NDAC Signal vom IEEE-Bus NDAC = Not Data accepted Bit 1 (Output): NRFD Signal zum IEEE-Bus NRFD = Not ready for data Bit 2 (Output): ATN Signal zum IEEE-Bus ATN = Attention Bit 3 (Output): Kassettenausgabe Bit 4 (Output): Motorkontrolle Rekorder #2 POKE 59456,239: Motor ein, 255: aus Bit 5 (Input): Display an (sync Signal) WAIT 59456,32,32 wartet auf Interrupt (alle 1/60 s) Bit 6 (Input): NRFD Signal vom IEEE-Bus NRFD = Not ready for data Bit 7 (Output): DAV Signal zum IEEE-Bus DAV = Data valid
E841	59457	SYNC, I/O Port A Bit 0-7 (I/O): User Port mit handshake
E842	59458	P2DB, Datenrichtungsregister Port B 0 = Input, 1 = Output
E843	59459	P2DA, Datenrichtungsregister Port A: 0 = Input, 1 = Output
E844	59460	T1L, Timer 1 low lesen, setzen
E845	59461	T1H, Timer 1 high lesen, setzen und Start
E846	59462	T1LL, Zugriff zu Timer 1 low
E847	59463	T1LH, Zugriff zu Timer 1 high
E848	59464	T2L, Timer 2 low lesen und IRQ-Flag loeschen Timer 2 low setzen und IRQ-Flag nicht loeschen POKE 59464,n für Musik über User Port (CB2)
E849	59465	T2H, Zugriff Timer 2 high und IRQ-Flag loeschen
E84A	59466	SR, Serielle Ein/Ausgabe mit Shift-Register (CB2) POKE 59466,n: n=15,51 oder 85 für Musik über User Port
E84B	59467	ACR, Hilfskontrollregister Bit 3 (I/O): Serielle Ausgabe vom User Port Setzt das Shift-Register in den 'free-running-mode' POKE 59467,16 für Musik vom User Port (CB2)
E84C	59468	PCR, Peripherie-Kontroll-Register Bit 3 (Output): Kleinbuchstaben oder Graphik POKE 59468,12: Grossbuchstaben und Graphik POKE 59468,14: Klein- und Grossbuchstaben
E84D	59469	IFR, Interrupt-Flag-Register
E84E	59470	IER, Interrupt-Enable-Register
E84F	59471	SYNC1, I/O Port A Bit 0-7 (I/O): User Port ohne Handshake

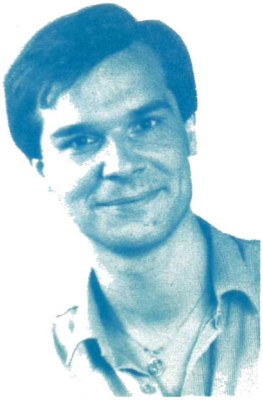
LABEL	HEX.	DEZ.	Routine bzw. Speicherstelle BASIC 4+3
	F000		Bytes für Fehler- und I/O-Meldungen
	F000	61440	'TOO MANY FILES'
	F00E	61454	'FILE OPEN'
	F017	61463	'FILE NOT OPEN'
	F024	61476	'FILE NOT FOUND'
	F032	61490	'SEARCHING'
	F03D	61501	'FOR'
	F041	61505	$0D 'PRESS PLAY'
	F04D	61517	'& RECORD'
	F056	61526	'ON TAPE #'
	F05F	61535	$0D 'LOAD'
	F064	61540	$0D 'WRITING'
	F06D	61549	$0D 'VERIFY'
	F074	61556	'DEVICE NOT PRESENT'
	F086	61574	'NOT INPUT FILE'
	F094	61588	'NOT OUTPUT FILE'
	F0A3	61603	$0D 'FOUND'
	F0AA	61610	$0D 'OK.'
	F0AE	61614	$0D 'READY.'

LABEL	3.0	4.0	Routine bzw. Speicherstelle
TALK	F0B6	F0D2	Sendet TALK Kommando zum IEEE-Bus
LISTN	F0BA	F0D5	Sendet LISTEN Kommando zum IEEE-Bus
ISOUR	F0EE	F109	Charakter auf IEEE-Bus ausgeben, Character in $A5
SECND	F128	F143	Sendet Sekundäradresse aus AC zum IEEE-Bus
SCATN	F12D	F148	Beendet ATN Zustand am IEEE-Bus
ERRPO	F136	F165	Fehlerbehandlung auf IEEE-Bus
ERRP7	F13D	F16C	
MSG	F156	F185	Ausgeben von Text aus dem Bereich F000-F0B5
TKSA	F164	F193	LISTEN Kommando und Sekundäradresse zum IEEE-Bus
CIOUT	F16F	F19E	Sendet Zeichen aus AC zum IEEE-Bus
UNTLK	F17F	F1AE	Sendet UNTALK Kommando zum IEEE-Bus
UNLSN	F183	F1B9	Sendet UNLISTEN Kommando zum IEEE-Bus
ACPTR	F18C	F1C0	Liest Zeichen vom IEEE-Bus in AC
GETIN	F1D1	F205	Eingabe eines Zeichens von der Tastatur, Einlesen eines Zeichens über den IEEE-Bus
BASIN	F1E1	F215	Input eines Zeichens
BSOUT	F232	F266	Zeichenausgabe Schirm, Kassette, IEEE-Bus
CLALL	F26E	F2A2	Anzahl der offenen Files = 0,(Filenummern, Primär- und Sekundäradressen bleiben erhalten) kein CLOSE !!!
CLRCH	F272	F2A6	Löscht IEEE-Kanäle, Rückkehr zur normalen I/O
	F284	F2C8	Setzt Input-Gerät = Tastatur($00), Output-Gerät = Bildschirm ($03)
JLTLK	F28D	F2C1	Sucht Eintrag in Tabelle der Filenummern
JZ100	F299	F2CD	Setzt Filenummer, Primär- und Sekundäradresse des laufenden Files, Uebertrag aus File-Tabelle in Variable ($D2-D4)
CLOSE	F2A9	F2DD	CLOSE-Befehl
	F2E5	F329	Entfernt einen File aus der Tabelle der logischen, primären und sekundären Adressen
STOP1	F301	F335	Test auf STOP-Taste; Z-Flag gesetzt, wenn Taste gedrückt ist
STOP	F30F	F343	Test auf STOP-Taste; Rückkehr zum READY-Mode, wenn Taste gedrückt ist
	F315		Test auf direktes Kommando zur Unterdrückung der Kommentare im Programm-Modus bei I/O (Kassette, Floppy-Disk) z.B.: LOADING, VERIFYING
TXTST	F31D	F351	Test auf Direkt-Modus
LD15	F322	F356	LOAD-Routine
LOAD	F3C2	F401	LOAD eines Programms
	F40F		'SEARCHING' oder 'SEARCHING FOR' ausgeben

LABEL	3.0	4.0	Routine bzw. Speicherstelle
	F41D		lfd. Filenamen auf Schirm (z.B. hinter WRITING) ausgeben
LD400	F42E	F46D	'LOADING' oder 'VERIFYING' ausgeben
PARS1	F43E	F47D	Holt Parameter für LOAD, SAVE, VERIFY Eingabe des Filenamens: SYS(62526)A$,Kassettennummer
	F466		OPEN IEEE-Kanal (für LOAD/SAVE Filenamen zum Bus)
	F47C		'DEVICE NOT PRESENT ERROR' ausgeben
FAF	F494	F4D3	Sucht nach speziellem File-header (Kassette)
VER	F4B7	F4F6	VERIFY-Routine
PARS2	F4CE	F50D	Holt Parameter für OPEN und CLOSE aus BASIC-Text
PR150	F516	F555	Ueberprüfung, ob lfd. Zeichen Komma, sonst 'SYNTAX ERROR'
	F519		Wenn nächstes Zeichen <> $00 (Zeilenende) und <> ':' (Befehlsende) wird 'SYNTAX ERROR' ausgegeben
OPEN	F521	F560	OPEN-Befehl
	F528		'FILE OPEN ERROR' ausgeben, wenn File vorhanden
	F55F		RECORD-SAVE-Routine
	F56E		'FILE NOT FOUND ERROR' ausgeben
	F570		Fehlermeldung aus Bereich ab $F000 ausgeben, Y-indiziert
SUFIHD	F5A6	F5E5	Sucht nach nächstem File-Header (Kassette)
TAPEH	F5DA	F619	Schreibt File-Header
	F5EB		Löscht Kassettenpuffer
LDAD2	F63C	F67B	Holt Start- und End-Adresse aus File-Header im Tape-Puffer und speichert sie nach $FB/FC bzw. $C9/CA
ZZZ	F656	F695	Setzt Startadresse Kasettenpuffer
LDAD1	F66C	F6AB	Setzt Start- und Endadresse SAVE für Kassettenpuffer
SYS	F684	F6C3	SYS-Befehl
SV60	F68D	F6CC	Tape-Start- und -End-Adresse setzen für SAVE BASIC-Programm
SAVE	F69E	F6DD	SAVE Befehl
SAVPRG	F6A4	F6D3	
		F6A4	Tape-Record-Routine
	F6A8		'DEVICE NOT PRESENT ERROR' ausgeben
CLSEI	F6F0	F72F	CLOSE IEEE-Kanal
CUDTIM	F729	F768	Uhrzeit erhöhen
CHKIN	F770	F7AF	Prüft und setzt Eingabegerät (Filenummer in XR)
CKOUT	F7BC	F7FE	Prüft und setzt Ausgabegerät (Filenummer in XR)
JTP20	F806	F84B	Tape-Kontroll-Routine, setzt Puffer-Anfangszeiger, erhöht Zeiger für Kassettenpuffer
CSTE1	F812	F857	'PRESS PLAY ON TAPE #x' ausgeben, wenn Rekorder-taste nicht gedrückt 'OK' ausgeben, wenn Taste gedrückt wurde
CS10	F835	F87A	Testet Tastenschalter von Rekorder, Z-Flag, wenn gedrückt
	F847		Wenn Tape-Taste gedrückt, Rücksprung zur aufrufenden Routine, sonst 'PRESS PLAY & RECORD ON TAPE #x' ausgeben und warten: 'OK' ausgeben, wenn Taste gedrückt wurde

LABEL	3.0	4.0	Routine bzw. Speicherstelle
	F855		Block vom Band in Tape-Puffer lesen (192 Zeichen)
BLCKR1	F85E	F8A3	Lesen eines Blockes von Band mit beliebiger Länge
	F886		Schreiben eines Blockes auf Band (192 Zeichen)
	F890		Schreiben eines Blockes auf Band (beliebige Länge)
	F89E		beendet Schreiben/Lesen
	F8E6		Testet Interrupt-Vektor HB auf $E6, wartet auf I/O-Abschluss
	F8F0		Testet Stop-Taste bei Kassettenoperationen, bricht ab, falls noetig und stellt alten IRQ ($E62E) wieder her
	F900		Initialisiert Timing für Kassetteneingabe
	F931		Interrupt-Routine für Kassetteneingabe
	FA57		Vom Tape eingelesene Zeichen abspeichern
	FB5B		Prüfsummentest
	FB76		Rettet SAVE-Adresse, speichert Zeiger auf letztes geschriebenes Zeichen
	FB7F		Setzt Statuswort ST
	FB84		Bereitet 8 bit-Zählung vor für Kassetten-I/O
	FB93		Schreibt ein Bit auf Kassette
	FBAF		Interrupt 2 beim Schreiben auf Kassette
	FC41		Interrupt 1 beim Schreiben auf Kassette
	FC7B		Abschluss Band-I/O, Tape-Motor aus, normalen IRQ-Vektor einsetzen, Bildschirm ein
	FC96		Interrupt-Vektor setzen, XR-indiziert
	FCA6		$3C nach $E813, ($E840)=($E840) OR $10, (Kassettenrekordermotoren aus)
	FCB4		Parityprüfung (über 8 Bit)
START	FCD1	FD16	Entry für Power on Reset, Cold Start
	FCFE		Entry für NMI (non maskable interrupt)
	FD01		Tabelle der Interrupt-Vektoren
CALLE	FD11	D472	Monitor (Call-Entry)
BRKE	FD17	D478	Monitor, Entry bei Software-Interrupt (Break-Entry)
	FD56	F4BA	Befehlseingabe-Routine des Monitors
SO	FD70	D4D4	Befehlsauswertung
PUTP	FD88	D4EC	Programmzähler aus $FB/FC laden
DN	FD93	D4F7	Anzeige von Speicherinhalten
BYTE	FDA7	D50B	1 Byte einlesen und abspeichern, (Zeiger $FB/FC)
SETR	FDBF	D523	Ausgabe/Eingabe von SR...SP vorbereiten
SPAC2	FDCA	D52E	Schreibt zwei Leerzeichen
SPACE	FDCD	D531	Schreibt ein Leerzeichen
CRLF	FDD0	D534	RETURN ausgeben
INCTMP	FDD5	D539	Pointer $FB/FC erhöhen, bei Nulldurchgang $DE erhöhen
CMDS	FDE0	D544	Befehlstabelle des Monitors
ADRH	FDE8	D54C	Sprungadressen-1 der Monitorroutinen
REGK	FDF8	D55C	Tabelle für Textausgabe des Monitors
ALTRIT	FE15	D579	Ausgaberoutine des Monitors für den Anfang jeder Zeile auf dem Bildschirm
DSPLYR	FE23	D587	Monitor: R-Kommando, Anzeige Registerinhalte
DSPLYM	FE58	D5BC	Monitor: M-Kommando, Anzeige Speicherinhalte
ALTR	FE97	D5FB	Monitor: ;-Kommando, Veränderung Registerinhalte
ALTM	FEB9	D61D	Monitor: :-Kommando, Veränderung Speicherinhalte
MGO	FECF	D633	Monitor: G-Kommando, Programmstart
EXIT	FF07	D66B	Monitor: X-Kommando, Verlassen des Monitors, BASIC-Warmstart

LABEL	3.0	4.0	Routine bzw. Speicherstelle
	FF11	D675	Monitor: L- und S-Kommando, LOAD und SAVE
	FFB1		CBM-Copyright-Statement
			Sprungvektoren
DOPEN		FF96	Sprungadresse: DOPEN
DCLOS		FF99	Sprungadresse: DCLOSE
RECORD		FF9C	Sprungadresse: RECORD
FORMAT		FF9F	Sprungadresse: HEADER
COLECT		FFA2	Sprungadresse: COLLECT
BACKUP		FFA5	Sprungadresse: BACKUP
DCOPY		FFA8	Sprungadresse: DCOPY
APPEND		FFAB	Sprungadresse: APPEND
DSAVE		FFAE	Sprungadresse: DSAVE
DLOAD		FFB1	Sprungadresse: DLOAD
DIRCAT		FFB4	Sprungadresse: DIRECTORY/CONCAT
RENAME		FFB7	Sprungadresse: RENAME
SCRATC		FFBA	Sprungadresse: SCRATCH
READDS		FFBD	Sprungadresse: Lese Disk-Status-Variable ds
COPEN	FFC0	FFC0	Sprungadresse: OPEN
CCLOS	FFC3	FFC3	Sprungadresse: CLOSE
COIN	FFC6	FFC6	Sprungadresse: setzt Eingabeeinheit
COOUT	FFC9	FFC9	Sprungadresse: setzt Ausgabeeinheit
CCCHN	FFCC	FFCC	Sprungadresse: setzt Standard I/O-Einheiten
CINCH	FFCF	FFCF	Sprungadresse: INPUT eines Zeichens
OUTCH	FFD2	FFD2	Sprungadresse: PRINT eines Zeichens
CLOAD	FFD5	FFD5	Sprungadresse: LOAD
CSAVE	FFD8	FFD8	Sprungadresse: SAVE
CVERF	FFDB	FFDB	Sprungadresse: VERIFY
CSYS	FFDE	FFDE	Sprungadresse: SYS
ISCNTC	FFE1	FFE1	Sprungadresse: testet STOP-Taste
CGETL	FFE4	FFE4	Sprungadresse: GET ein Zeichen
CCALL	FFE7	FFE7	Sprungadresse: Abbruch aller I/O-Kanäle
	FFEA		Sprungadresse: Uhrzeit erhöhen
	FFFA	FFFA	Vektor für NMI, zeigt auf $FCFE<FD49
	FFFC	FFFC	Vektor für RESET,zeigt auf $FCD1<FD16
	FFFE	FFFE	Interrupt-Vektor IRQ und BRK, zeigt auf $E61B<E442



Hans Dieter Winter
Dipl. Ing. (FH) Informationstechnik

Als Grundlage dieses Buches diente die «Analyse des Betriebssystems des CBM 3000», die 1981 in Zusammenarbeit des Autors mit seinem damaligen Studienkollegen und jetzigen Dipl. Ing. (FH) Gerald Eglmeier entstand. Es baut ferner auf einigen bisher erschienenen Veröffentlichungen zum Commodore-Computer auf. Darüberhinaus werden sämtliche «Spezialitäten» der CBM-Betriebssysteme ausführlich erläutert und für den Assemblerprogrammierer durchsichtig dargestellt. Das so entstandene Werk entspricht etwa einem ausführlichen CBM-Handbuch für jeden Programmierer, der in Maschinensprache mit dem CBM-Rechner arbeiten will.

Durch Überarbeitung der BASIC 3.0-Version und Erweiterung auf die CBM 8000-Serie mit sämtlichen neuen Adressen, Funktionen und Möglichkeiten sowie Einführung der üblichen Labels wurde die vorliegende Analyse von Dipl. Ing. (FH) Hans-Dieter Winter erarbeitet.

ISBN 3-907007-01-8